Motion Control PMC

PMCprimo

Reference Manual – Item No. 21 507-07

1 General Information

1.1 Copyright

Copyright 2005 Pilz GmbH & Co. KG. All rights reserved.

All rights reserved. The implementation of technical changes which improve the performance of the product is subject to change without prior notification! The product or parts of the content may be reproduced or transmitted in any form (by printing, photocopying, microfilm or any other method) or stored, processed, copied or distributed by electronic means without the written permission of Pilz GmbH & Co. KG.

1.2 Notice

All examples in this manual have been made to ensure the accuracy of the information in this document.

Pilz GmbH & Co. KG assumes no liability for inadvertent errors. Suggestions for improvements in either the products or the documentation are welcome.

1.3 Previously published editions

Previous Versions:

Edition	Comment
PH.V2.0.GB	For including unless PMCprimo version 1.006
V3-11-11-2002	For including unless PMCprimo version 1.008
V3-15-05-2003	For including unless PMCprimo version 2.000
V3-15-06-2004	For including unless PMCprimo version 2.004
V4-23-07-2004	For including unless PMCprimo version 2.004
V5-03-02-2005	For including unless PMCprimo version 2.005
V6-15-06-2005	For including unless PMCprimo version 2.006
V7-04-11-2005	For including unless PMCprimo version 2.007

2 Contents

1 General Information 2

1.1 Copyright 2
1.2 Notice 2
1.3 Previously published editions 2

2 Contents 3

3 Abbreviations and Symbols 6

4 Introduction 7

5 General Description 8

6 News in PMCprimo firmware version 2.006 9

7 Commands 11

7.1 General Notes 11
7.2 Command Execution 14

8 Command Reference 16

8.1 Data Commands 16
8.2 Mode commands 27
8.3 Move commands 28
8.4 Parameter commands 38
8.5 Sequence commands 52
8.6 Map commands 66
8.7 Wait command 87
8.8 Error Trapping 93
8.9 Gain commands 98
8.10 Reference Commands 104
8.11 Digital Inputs and Outputs 122
8.12 Configuration Commands 135
8.13 Display Commands 161
8.14 Phase Advance 170
8.15 Analogue Control 174
8.16 Variable commands 179
8.17 Commands for CANopen network 189

9 Status- and error messages 191

9.1 Status messages 191
9.2 Error messages 192
9.3 Status Codes 200

10 Interface PMCprimo <-> machine 201

10.1 Notes on Installation 201
10.2 Safety 202
10.3 Position Encoder 202
10.4 Demand Output (PMCprimo 16+) 203

10.5 Auxiliary Output 203
10.6 Serial Communications 204
10.7 CANopen Interface 204
10.8 Analogue Inputs 204
10.9 Digital In- and Outputs 205
10.10 Drive commands 207
10.11 Operate modes 209
10.11.1 Host 209
10.11.2 Node 209
10.11.3 CAN-Net and CAN-Open 210

List of Figures

Figure 1: Trapezoidal profile 28
Figure 2: Position profile with trapezoidal velocity profile 29
Figure 3: Triangular move profile 30
Figure 4: Move with normal stop 32
Figure 5: Move with abort 33
Figure 6: Constant velocity move 34
Figure 7: Initialisation to zero position 35
Figure 8: Initialise position and bounds 36
Figure 9: Move with change of velocity 38
Figure 10: High and low acceleration (SA = DC) 39
Figure 11: Effect of DC Command 40
Figure 12: Move with creep to final position 42
Figure 13: Normal/slow velocity mode 43
Figure 14: Timeout for window TI 44
Figure 15: Simple position maps 66
Figure 16: Position mapping over a defined range 67
Figure 17: Position map for a cyclic machine 68
Figure 18: Effects of map base and map offset 69
Figure 19: Example setup for software differential 70
Figure 20: Example setup of software differential (complex) 70
Figure 21: Position mapping as an equation 71
Figure 22: Position mapping as an equation (differential) 71
Figure 23: Software clutch 85
Figure 24: Wait time 87
Figure 25: Wait for input line 88
Figure 26: Wait for absolute position 89
Figure 27: Wait for relative position 90
Figure 28: Monitor output functions (this diagram is not to scale) 102
Figure 29: Cooperation of the commands after detecting a reference signal. 105
Figure 30: Cooperation of the commands reference error correction 106
Figure 31: Position bounds 108
Figure 32: Reference correction with “RV” 113
Figure 33: Reference correction with “RV” 114
Figure 34: Reference width checking with ZH, ZL, FH and FL 117
Figure 35: Mask input with MI command 127
Figure 36: Inhibit input function 128
Figure 37: Example 1 for command PO 144
Figure 38: Example 2 for command PO unless version 1.006 145
Figure 39: Example 3 for command PO since version 1.007 145
Figure 40: Phase advance 171

3 Abbreviations and Symbols

	Meaning
PMCprimo	Registered trademark of Pilz GmbH & Co. KG
PMCtendo	Registered trademark of Pilz GmbH & Co. KG

Symbol	Description
	This symbol indicates a possible danger, hazard, risk to life or health. Ignorance may seriously affect health and cause dangerous injuries.
	This symbol indicates an example.
	This symbol indicates special user tips and/or important useful information. These will support optimum use of the product and functions.
•	Emphasis indicator

4 Introduction

This manual describes the complete functional range, which is available for the programming in PMCprimo-motion control-systems.

For programming PMCprimo-systems are equipped with a RS-232 serial interface, and depending on model additionally with an Ethernet interface. With these interfaces PMCprimo can be connected with a PC. The communication is done with a terminal software. Easy to learn commands allow effective and fast programming and initial operation.

Alternatively to the described programming language, it is possible to completely program PMCprimo-motion control-systems with a IEC 61131-3 (CoDeSys) software. The functions and commands of this programming language is described in a separate manual (AH-PMCSLC-xx-GB).

Please read the manual carefully. Reference manual for all commands is the german version.

PMCprimo-motion control-systems resolve tasks with the help of electronic axis synchronisation, when applied correctly, which where so far realised by mechanical components. For these tasks electronic cams, cam gearboxes and other functions are used to control servo power transmissions highly dynamically.

Additionally PMCprimo-motion control-systems have the possibility to take over the complete control of the machine with a Soft-PLC.

Prerequisite for this is the comprehension of the system basics. This manual efforts to create the prerequisite for this. Another effective way is a training at **Pilz**.

5 General Description

This section gives a brief description of the facilities of the PMCprimo-motion control-systems.

PMCprimo-motion control-systems are available in different types and can control more than 700 axes in a network. The coordination of the separate axes in PMCprimo is realized over the host level. Every PMCprimo-device can be used as a host system.

PMCprimo is programmed in a high level language or IEC 61131-3 (CoDeSys). Central communication medium is a terminal software (see user manual “PMCprimo Motion Control Tools”) on your PC, that will be connected via a RS232 or Ethernet interface. All parameterisations and user programs can be saved in flash memory or a Compact-Flash-Card.

Software functions for movement of servo axes

With the described PMCprimo-commands the parameters will be set and functions for the respective task will be programmed.

The wide range of commands allows the realization of complex applications. PMCprimo has axis spanned commands, this means that a separate command can affect several axes.

Reference commands allow the automatic initialisation and a reference error correction at a running machine. The reference error can be compensated variable, for example with defined ramp or time functions.

By this, divergences in product or machine parameters can be corrected flexible (cycle based).

Free definable master-slave-relationships allow the user to define any speed profiles for motors. The usage of jerk less movement flow (for example modified sine) reduces the mechanical load at the machine as much as possible.

PMCprimo can fit motors with electronic gear functions just by software. The relation between motor positions can be defined freely by the user, additionally to the electronic gear function (linear and non linear correlation with a tabular position assignment). There are multiple of VDI defined movement kinds for your choice. Tabular position assignments can replace mechanical cams and mechanical gearboxes.

When needed PMCprimo can generate new maps with the internal motion generator during machine operation. Product dependent machine changeovers can be done by the push of a button.

Beside several possibilities to synchronize machine axes, there are also many commands for absolute, relative or endless positioning of power transmissions available.

All axes can operate in virtual mode (electronic master axis), for example for initial operation or as help axes.

Hardware functions

PMCprimo is intended for use with digital incremental position encoders which provide two signals in
quadrature. This allows the system to measure both the distance and direction of motion of the motor, thus providing the closed-loop feedback information for the channel. The encoder input interface circuit multiplies the resolution of the encoder by four, such that each complete cycle of the encoder signals represents four counts. PMCprimo includes full isolation of the encoder input signals, and are designed for use with encoders having differential line driver outputs. This is get best performance and noise rejection in an industrial environment. You can also use SSI-, Hiperface-, or CANopen-encoder instead of incremental position encoders.

PMCprimo has digital input and digital output lines (the number depends to the type of PMCprimo), which may be used in various ways. Additional has PMCprimo analogue inputs and outputs. All digital input lines gives with a change of their state an interrupt. The user defines what happens with the interrupt. The state of the input line or the value of the analogue input could be checked in sequences. The state of the outputs is also controlled by sequences. Outputs may be explicitly set and cleared, and can be used to control external relays or valves, or just for status indication.

The inputs and outputs can be used as an interface to a PLC The digital input and output lines are fully isolated and are compatible with 24 V logic signals. For comfortable communication with a PLC or any other host computer a RS232 interface, MODBUS, several field bus systems or Ethernet are available.

The analogue outputs give signals, ranging from -10 V to + 10 V.

Applications:

Typical applications for PMCprimo-motion control-systems are:

- Packaging machinery
- Modular machines
- Printing and paper machineries
- Handling systems

6 News in PMCprimo firmware

6.1 Version 2.006

Command:	News/Changes	Page:
MT	New range for MT: 0 to 1000 milliseconds	86
RK	New range for RK:	120
MW	Alignment acceleration also with speed mapping	81
AO	Assignment to variable now possible	103
DP	Actual position with demand positionover CAN	161
AV	Higher resolution for AV setting	79
DL	Define of direction	136
PW	Files are encoded when saved on CF card	21
SU	Display “uPd” while updating firmware	23

6.2 Version 2.007

Command:	News/Changes	Page:
LL	The default value for LL was changed to –4.194.304. Otherwise the channel shows an error message while moving because of SB default value of 4.104.304.	95
LH	The default value for LH was changed to 4.194.304. Otherwise the channel shows an error message while moving because of SB default value of 4.104.304.	94
Drive commands	Floating point numbers will be corrected command dependant.	207
	The error message “Attempt to divide by zero” now also shows the program and line number. Now it is possible to find the exact program line which caused this error.	
CD 3	Additional CAN baudrate settings	16
LR	The message “Reference error reduced to maximum value” not shown if LR != 0.	112
TW	New setting for TW command	165
ZW	Bit 5 of ZW controls whether “**SB**“ has effects for the command “**WA**”.	50

7 Commands

7.1 General Notes

All commands are two letters, sometimes followed by a numerical parameter. The syntax of this commands is described below. Numerical parameters are denoted by "n", "nn" or "m". Parameters entered as a binary string (‘0’s and ‘1’s) are denoted by "bb". All input commands or command strings are terminated by a carriage return <CR>. Commands are given together with the numerical parameter to the system.

PMCprimo sends after a RETURN a prompt back. This prompt includes follow informations:

```
0.1:
```

0	.	1	:
actual node. In configuration "STANDALONE", it is always '0'.	separator	actual channel	actual state (see page 191).

Commands without a numerical parameter show the actual value.

Example 1 (command with numerical parameter):

```
SV2000 <CR>
```

Example 2 (command itself):

```
SV<CR>
SV=2000
```

Numerical parameters are given decimal or hexadecimal to the system. Decimal numbers are entered as signed or unsigned (assumed positive) numbers. For entering hexadecimal you must prefix "0x". Negative hexadecimal number have a minus-sign.

Leading zeros may be omitted when entering values.

The normal character set consists of the letters „A-Z“ and „a-z“, the numbers „0-9“ and "_", and space characters. Commands may be sent in either lower case or upper case. Lower case commands are converted to upper case before storing as command sequences or function input strings.

Multiple command strings may be entered as one command line, with the individual commands separated by a delimiter character. The ‘;’ character must be used as the command delimiter.

Example:

```
0.1>ch2;mr2000;wt500<CR>
```

The delimiter between the command strings is the ‘;’-sign. The maximum line length is 200 characters. The backspace character is used to remove characters from the current input line. Other non-printing characters are simply echoed, and have no effect. The escape button is able to stop a printing command, or the „??“-command („??“-command see page 168) for a help listing.

Any '+' characters at the beginning of an input line are ignored. This is to prevent any errors when using a modem link to a remote system. The standard hangup character sequence is "+++", and this would otherwise leave the system with tw0 '+' characters in the input buffer after the modem link is closed.

The standard command set provides flexible and complete control of PMCprimo. The commands fall broadly onto the following categories.

- Data commands (chapter 7.1)
  Commands to change between channels and to handle the stored setup data.

- Mode commands (chapter 7.2)
  These include commands to change between motor off and position control.

- Move commands (chapter 7.3)

  These are the basic commands for moving and stopping the motors, using the normal trapezoidal or S-form move profile..

- Set parameter commands (chapter 7.4)
  These commands set up a wide range of parameters, including the velocity and acceleration of the normal moves.

- Sequence commands (chapter 7.5)
  These commands allow the user to enter, list and execute complex command sequences.

- Map commands (chapter 7.6)
  These commands allow the user to enter, list, and execute position mappings (Software Gearbox).

- Wait command (chapter 7.7)

  These commands are used in command sequences to wait until a condition is true before executing the next command in the sequence.

- Error handling (chapter 7.8)
  These commands set up the error monitoring functions and defines the reaction of PMCprimo if an error appears.

- Gain commands (chapter 7.9)
  These commands set up the gain constants used in the closed-loop control algorithm.

- Reference commands (chapter 7.10)
  These commands set up the continuous position correction facilities for use with position reference input signals.

- Digital input and output commands (chapter 7.11)

  These commands directly control the digital input and output lines.

- Configuration commands (chapter 7.12)
  These commands configure the digital input and output lines for various automatic functions.

- Display commands (chapter 7.13)

These command outputs parameter values and status information via the serial port.

- Phase advance commands (chapter 7.14)
  These commands control a speed-dependent phase advance mechanism for slave channels in mapping, and for position trigger outputs.

- Analogue control commands (chapter 7.15)
  These commands are used to set up closed loop tension control.

- Variables (chapter 7.16)

  Variables and expressions can be used in place of constants for most command parameters to increase the flexibility of PMCprimo. They are of particular use in conjunction with an operator's panel or a bus system.

In PMCprimo 2+2 and 16+ (PMCprimo Drive2 optional) it is possible to save variable values in the battery buffered RAM. These values are available after turning the device on and off. The actual encoder value is also saved in the RAM.

The PMCprimo 16+ (PMCprimo Drive2 optional) has a CompactFlash card on which all values can be saved additionally.

The input of drive parameters for the PMCprimo Drive/2 can be done directly with PDrive or PTerm.

All position values occur from encoder impulses (increments), Hiperface or SSI.

Data and parameter values are saved in a flash memory with the command "**SP**". For the saving of programs 2 MB memory are available. Variables are also saved dynamically in this memory area.

A checksum test checks the saved data during the booting of the control. If an error is found, an error message is given and the saved values will be deleted and the default values are loaded. An error may appear for example if the PMCprimo is turned off during saving.

7.2 Command Execution

PMCprimo commands can be executed in a number of different ways. This section explains how PMCprimo deals with different methods of execution and how to get the most out of your system. The main ways of executing commands are as follows.

Command line

Commands can be entered separate or as a string at the RS232 terminal and they are executed immediately when <CR> is typed.

Entering a command line each command in the line is executed step by step. It is possible to execute command lines parallel.

Example for command line:

`0.1>MA1000<CR>`	PMCprimo moves the motor to the target position of 1000 increments

Example for command line:

0.1>MA1000;WT50;MA0<CR>	PMCprimo moves the motor to the target position 1000, waits 50 ms and moves the motor to target position of 0.

Sequences

Sequences containing one or more lines of commands can be defined using the "**ES**" and "**NS**" command. The commands are executed by issuing the "**XS**" command. Sequences can themselves call other sequences. The sequences can run parallel started with the command "**XP**", trigger-variables or inputs. A sequence can not be started if it is in use.

Example:

```
0.1> ES PROG1<CR>                          Definition of sequence PROG1
002E CH1;MA1500;VC+;WT100;ST<CR>
003E NS <CR>                               end PROG1
0.1>

0.1> ES PROG2<CR>                          Definition of sequence PROG2
002E CH1;DP;DV<CR>
003E NS<CR>                                end PROG2
0.1>

0.1> XS PROG1<CR>                          Start sequence PROG1
0.1> XS PROG2<CR>                          Start sequence PROG2

0.1> DP=345                                Output of actual position of channel 1
0.1> DV=0                                  Output of velocity
```

Programs are itemized in sub programs, which will be started on different axes. If only axis specific commands are used in a program, then the program can run autonomous on an axis without a host. With these node programs the reaction time of input functions can reduced dramatically.

Input line function

Digital input lines are calling sequences. The relation between input line and sequence is defined with the "**DI**" command.

Trigger variable

To write a value to a variable triggers a sequence. The value of the variable must not change for executing the trigger. Trigger variables are working in the same way as the input lines and is subject to the same rules.

Sequences are automatically split up into a number of sub sequences which are executed on the individual channels.

Example for trigger variable:

`0.1>$TRIGVAR>TEST`	The variable TRIGVAR is defined as a trigger variable.
`0.1>EV $TRIGVAR`	The trigger variable is enabled.
`0.1>$TRIGVAR=0`	The sequence TEST is started with the variable assignment.

8 Command Reference

8.1 Data Commands

Commands for data management and system configuration of PMCprimo.

CD Change the operating mode

(**C**hange Operating Mo**d**e)

The command defines the operating mode of PMCprimo: The following operating modes are available:

- **STANDALONE**
  PMCprimo without network (standard).
- **HOST+NODE**
  If PMCprimo is used in a network, one network member must be set to "**Host + Node**".
- **NODE**
  If PMCprimo is used in a network, for every network member this operating mode must be set (except node 0 Host + Node)

Example:

```
0.1: cd <CR>
0.1:
Actual configuration:
Operate Mode: HOST+ACHSE
Change Operating Mode? Y/N
<0> STANDALONE
<1> HOST
<2> SLAVE
<3> HOST+SLAVE
Enter Operating Mode: 0 <CR>
Please reboot system
```

Data will be lost if you change the operating mode!

It is also possible to delete the PMCprimo program with this function.

Example:

```
0.1: cd
0.1:
Actual configuration:
Operate Mode: HOST+ACHSE
Change Operating Mode? Y/N: N<CR>
Delete application data? Y/N: Y><CR>
Are you sure? Y/N: Y<CR>
Reset is done
0.1:
```

For PMCprimo 2+2/16+ (PMCprimo Drive2 optional) the IP-address for Ethernet and the field bus address can be set.

Enhancement as of **version 1.008**:

For PMCprimo 2+2/16+ (PMCprimo Drive2 optional) it can set an offset for the field bus address. The range of value is between 0 and 100. Default is 0. Therefore it is possible to use the input ports 0.101 to 0.108 with the bus variables $B101 to $B108.

Example:

```
Actual configuration:
Operate Mode: STANDALONE
Actual IP address 10.10.180.101
Actual Netmask 255.0.0.0
Fieldbus In/Out length 50 words
Change Operating Mode? Y/N: N<CR>
Delete application data? Y/N: N<CR>
Change Ethernet? Y/N: Y<CR>
New IP address (00.00.000.000) ? 10.10.180.101<CR>
New Netmask (255.0.0.0) ? 255.0.0.0<CR>
Change Fieldbus configuration? Y/N: N<CR>
0.1:
```

Enhancement as of **version 1.008c**:

The size of the flash memory for PMCprimo an the SoftSPS can be set. Up to now the 256KB are divided into 128KB for PMCprimo and 128KB for the SoftSPS. Now the SoftSPS can get 128, 192 or 256KB. The flash for PMCprimo is 128, 64 or 0KB if a SoftSPS is installed.

Enhancement as of **version 2.000**:
The command was expanded with 2 optional parameters: **CD [n,x]**

Range n:

As of **version 2.005**:	2	All values of operating mode are set to default values: IP address: 192.168.0.1 Netmask: 255.255.255.0 Gateway will be deleted
	3	CAN baud rate 500 kBit or 1Mbit
	5	Number word field bus
	6	Offset field bus
	7	PMCprimo 2+2 fifth channel on or off
	8	cycle time CAN in standalone 1,2 or 4 ms
	9	CAN node number for Standalone
	10	Segmentation of memory for PMCprimo und SoftSPS
	11	Profibus address for Anybus-IC
As of **version 2.005**:	12	Set number of axis. With x=0 the default value (10) is set.
	14:	Configuration of RS232 interface
	15:	Configuration of RS422 interface

Range x: depends on n

With parameters it is possible to make settings in a sequence without the menu. The actual configuration is given out by the command “**LA**” and “**SP**” with “**CD**” commands. A backup is made complete in this way.

In operate mode standalone it is possible to set the node address for the CAN bus. With this it is possible us use more than one PMCprimo devices in one CAN net. For example one CAN operator panel can respond to several PMCprimo devices.

Enhancement as of **version 2.000**:

The cycle time of the can bus can be changed. Possible settings are 1, 2 and 4 (default) milliseconds. The setting is only in standalone mode possible.

It is important to use the PMCtendo DD4 version 3.56 with using 1ms, because the parameterization of the drive is not possible with an older version.

Enhancement of **version 2.000** for PMCprimo 2+2 and 16+:
Change RS-232 configuration:
It can choose between software- (Xon/Xoff) and hardware-handshake.

Enhancement of **version 2.000** for PMCprimo 16+:

The date and time of the real time clock can be set. Furthermore the number of channels can be set. It depends on the operate mode and the CAN-card:

Host+ Node and **Standalone**:	Number of channels can be set between 1 and 20. Default: 10 Without a CAN-card all axis above number 3 are virtual. With CAN-card all axis above number 16 are virtual.
Node:	Number of channels is set automatic. It can not be changed. If a CAN-card is not inside, it exist 10 axis (8 real and 2 virtual) With a CAN-card, it exist 18 axis (16 real and 2 virtual).

RS-422 configuration for PMCprimo 16+:
The modbus interface can set to tristate (multidrop).

Enhancement of **version 2.003d**:

There is no configuration with the "**CD**" command. The 7 segment display can be switched to "**run**" if no error is active with "**CD17,1**". With "**CD17,0**" the old display is activated.

Enhancement of **version 2.004:**

Die 7 segment display can be switched with "**CD17,1**". When this was done, the display just shows "**run**", if no error occurs. With "**CD17,0**" the previous display is activated.

Enhancement of **version 2.005:**

With option 4 it is possible to set a gateway address.

Enhancement of **version 2.007:**

For option 3 additional CAN baud rate settings were added:
Now this settings are possible:
20, 50, 100, 125, 250, 500, 800 and 1000 kBit/s.

VN Print version number

This command prints information about the version of software fitted to PMCprimo. It gives the version number and revision dates of the firmware.

Enhancement as of **version 2.000**:
This command can set to a variable. Follow information is set node specific to the variable:

Hardware	Version-No.	Version-No. SubIndex	Version-No-SubSubIndex	Supplies
Nibble	Nibble	Byte	Byte	Nibble

Hardware: 2: PMCprimo Drive
3: PMCprimo 2+2
4: PMCprimo 16+
5: PMCprimo Drive2

Version-No.: Numerals of version number before the comma
Version-No. SubIndex: Numerals of version number after the comma
Version-No. SubSubIndex: If beta version, 0xBn, with e.g. n = 1 for Beta1 resp. a for Version 1.008a
Supplies: Bit 0: Motion
Bit 1: Ethernet
Bit 2: SoftSPS

Enhancement as of **version 2.005**:
The command prints also the serial number

```
0.1:VN
0.1 Primo motion control 16+ Channels 10, Version 2.005 Nov 26 2004, 12:07:32, Ser.Nr.:4
```

HW Print hardware

Command as of **version 2.000**

This command prints information about the hardware in all CAN net ware which is no PMCprimo.

PR'Name' Print to terminal

Command as of **version 1.008**.

range 'Name': string with max. 60 signs.

This command prints the String 'Name' to a connected terminal. It is possible to make 100 PR commands in programs.

Enhancement as of **version 2.005**:

It is possible to make 200 PR commands in programs

MUnn Set modbus number

Range: 0 to 255
Default: 1

This command sets the modbus number. This is used in a ModBus multidrop net, to indentify the device.

BMnn Set baud rate modbus

Range: 9600 to 38400
Default: 9600

The baud rate for the modbus can be set. As the baud rate of 19200 Xon/XOff is deactivated.

BDnn Set baud rate

(Set **b**aud rate)

Range: 9600 to 115200
Default: 9600

The baud rate for the serial interface RS232 can be set. The parameter can't be saved. For PMCprimo Drive/2 only the default baud rate 9600 is possible.

PW Set password

This command allows the user to define his own password. The maximum length of the password is 8 letters. The password is saved with the "**SP**" command (see Page 22) inside the flash memory. Once a password is given to PMCprimo, PMCprimo asks for the password at the end of power up. If the entered password is not correct, there is no communication about the serial port. All the other functions are working as normal.

As of **version 2.006**:
If the password is set with the command PW then the files primoprg.pg and variable.prg are encoded on the CF card. Therefore nobody can use a CF cardreader and open the files without permission.

Example:

```
0.1>PW Name<CR>          enter password
0.1>
```

SP[n] Save parameters

n = 0: save all (can leaved out)

n = 1: save the program-datas (no variables).

n = 2: save only variables.

n = 4: Data selected with command '**LA4**' in Motorola s-record format can read with this option. It is possible to save a program for a network without nodes in the host. PMCprimo must be in the right operating mode before download because the change of mode deletes the flash.
Enhancement as of **version 2.000**:
If a SoftPLC is present, data of it are also read.

This command saves all the programmable parameters to flash memory.

During the saving, do not switch off the control. Otherwise data may be lost.

Only the variables defined in a program are saved.

There may be a delay while the save operation takes place, depending on the number of channels and the amount of data to be saved.

At the end of the save operation, PMCprimo calculates a checksum on the saved data by means of a cyclic redundancy check (CRC) algorithm. The checksum is then also saved in flash memory. This checksum ensures, that the data was read in correctly during booting.

While the save command is executed, no sequence is working. To make sure that everything is correct save the data in the way described below.

`GX`	Stop all sequences
`BV;BI`	Mask all trigger variables and input lines
`GF`	All motors off
`SP`	Save all data
`WT100;EV;EI`	Wait 100-ms and enable trigger variables and input lines.

The used commands are described later in this manual.

In a PMCprimo Drive/2 also drive-parameters are saved in a sequence (see page 180).

Enhancement as of **version 2.000**:

When program data (not variables) are saved a version number is increased and saved in the flash. Sequence changes made by third can retraced with this version number.

With the command “**LA**” (page 24) this number (#Version: XX) is displayed. Are only the variables or is the flash memory selected (command “**LA2**” and “**LA4**”) the version number is not displayed. If the program is deleted with “**CD**” or by the boot menu the version counter is set to zero.

Enhancement for PMCprimo 16+:

A backup is automatic made on a Compact Flash Card.

SU Software Update

With this command it is possible to update a new firmware. After enter the command a control request appears. Reconfirming it, the file with the “Motorola S Record format” can download with the terminal program PTerm. The file has the extension "*.hex". With the RS232 interface the update needs a lot of time (about 1 hour). It is better to do the update with Ethernet connection if possible. With PMCprimo Drive it is not possible to update the firmware with Ethernet. With PMCprimo Drive2 update with Ethernet is possible, if you have the expansion board 1 expansion card.

It must made a reboot to activate the new firmware.

Enhancement as of PMCprimo 16+:
It is possible to start a software update from Compact Flash. The available versions are displayed and it can choose a specific version. Optional all version of nodes can updated.
It is also possible to update the PMCtendo DD4 firmware.

Example:

```
0.1: su
0.1:

Update firmware: (1=Yes/0=No/?=help) ?

Please download HEX-file now and follow the further
instructions!

12686Programming...
 Don't switch off now !!!

0.0 Softwareupdate successful (00:22:44)
0.1:
```

Enhancement as of **version 2.005**:

With SU command you can alsa make a update with CAN bus. Till now this was only possible to do by the boot menue.
Using a parameter the follow updates are possible:
SU0: The same update like without parameter
SU1: Firmwareupdate PMCprimo of all MC-drives
SU2: Firmwareupdate PMCprimo of all MC2+2
SU3: Firmwareupdate PMCprimo of all MC16+
SU4: Firmwareupdate PMCprimo of all MC-drive2
SU5: Firmwareupdate PMCtendo of all DD4
SU6: Firmwareupdate PMCtendo of all MC-drive2

Enhancement as of **version 2.006**:

If a firmware is updated with “**SU**” or the autoupdate function then the host shows "uPd" in the display.

RS[n] Reset to default setup

This command resets all the parameters, input and output line definitions, sequences and profiles to their default settings. On power-up, the system recalculates the checksum on the saved data in the flash memory. If the calculated checksum does not match with the stored checksum, the "**RS**" function is executed automatically to reset the system to the default state.

Enhancement as of **version 2.000**:
With the new paramter n:
n=0: Classic function described before.
n=1: A complete reboot of the control is made.

LA[n] List all parameters

n = 0: list all (can leaved out)

n = 1: list the program-data (no variables)

n = 2: list only the variables

n = 3: list all from flash

n = 4 List all in Motorola S-Record Format. It is possible to save a program for a network without nodes in the host with the command 'SP4'.

This commands lists all the parameters of PMCprimo, input and output line definitions, sequences and maps to the screen in a suitable format for entering the parameters etc. at a later state.

The escape-key interrupts the LA-command.

The command with n = 4 lists the data from the flash in a special format (S-Record). It is possible to save this data for a network without nodes in the host with the command '**SP4**'.

Using a PMCprimo Drive/2 the drive parameters are listed as well. They are defined in a sequence with a special name:

Name of the program: DRIVE_n
whereat n: Number of node

Enhancement as of **version 2.000**:
If a SoftPLC is present, with n = 4 the data of it are also listed.

If a PMCtendo DD4 is in the CAN net the drive parameters are listed as well. They are defined in a sequence with a special name:
TENDO_n: DD4 with CAN address n in CAN-Net
TENDO_n_m: DD4 with CAN address m in CANOpen of node n.

FM Display free memory

This command displays information about the memory. The information refers always to the host.

Example:

```
0.1: fm <CR>
0.1:
938904 bytes ram memory free
123004 bytes flash memory free
```

ZT[Name] Display active programs

Without any program name the command lists all active sequences.

Enhancement as of **version 2.000**:
The command line of a online command is also write out.

Example1:

PMCprimo User	Comment
`0.1: zt`	Set command
`0.1:`	
`Aktive processe:`	display active programs:
`==========`	
`I0.1:3 => PRG1`	Program PRG1 is active and was started from input 0.1:3.
`ONLINE(500) => PRG3`	The program PRG3 was started from a single command of the user
`$VAR1 => PRG2`	The program PRG2 was started from trigger variable $VAR1.
`ONLINE(501)`	Online-command "**ZT**" has the intern program number 501
`$VAR2 => PRG4 => PRG5`	The program PRG5 was called by PRG4. The program The program PRG4 was called by the trigger variable $VAR2.
`0.1:`	

IF "**ZT**" is followed by a program name, the active program line is listed if the program is active.

Example 2:

PMCprimo User	Comment
`0.1: ZTPRG2`	Command "**ZT**" with program-name
`0.1:`	
`Active processes: <PRG2>`	`node: 0 line: 2` The active program-part is running on the host (node 0), at the moment line number 2 of PRG2 is active.

SK Set software license key

New software keys can be set and shown with this command. The following software options are available only at additional charge:

- Motion Generator
- Soft-PLC

In devices, which where ordered with a special option, the according software keys are already unlocked.

The command “**SK**” shows the serial number of the device and the enabled options. It enables to enter a software key or to lock the actual options. To upgrade a software option you can order a software key and unlock it with “**SK**”.

To lock an option, the software key must enter a second time.

This example deletes a software key:

```
0.1:
0.1: sk<CR>
0.1: Serial number: 000164
Installed Softwarekeys:
Motion: b976c5

New Key?   b976c5<CR>
Key: Installation successful!<CR>
0.1:
```

8.2 Mode commands

PC Enter position control mode

This command activates the position control mode of the motor.

This command puts the current motor channel back into the normal state with the motor position continuously controlled, after the "**MO**" (motor off command) has been executed or any motor error has occurred. The prompt character '>' is returned to the terminal program (e.g. Pterm) in position control mode.

With active position control mode in PMCprimo 2+2 and 16+ in the first two channels a relay is switched over for use a drive enable signal if required.

MO Motor off

Turns off the position control servo loop action. All other facilities still operate normally, including the input and output lines, and the encoder position is continuously monitored. When the system is returned to position control mode, the motor does not jump back to the last position, but remains the new position. The system returns a ':' character as a prompt in the motor off state.

In the motor off state with PMCprimo 2+2/16+ the signal output is switched directly to 0V by the onboard relay. It is recommended that this relay is used to disable the motor drive completely. If the drive is not disabled with the relay, the motor will drift because the motor position is not controlled in this state.

If the "**MO**" command is used as a motor off stop command when PMCprimo puts the current channel into motor off and leaves the remainder of the sequence running. If it is necessary to put all channels into motor off then the "**GF**" global motor off command should be used.

VMn Set virtual motor mode

Range: 0 to 1

This command defines whether the current channel is in normal or virtual motor mode. In virtual motor mode, the channel can operate without a motor or encoder connected. The actual position is internally calculated from the demand position.

Virtual motor mode is particularly useful in the following circumstances.

- For testing commands and sequences before the controller is connected to the machine.
- For providing a dummy master axis in position mapping. For example, two axes could be made to trace a circular outline by providing a third master axis moving at constant velocity in virtual motor mode and mapping the two slave axes to the master with a sine and cosine map respectively.

VM0: The channel is set to normal mode. A real motor is controlled and a real encoder is read for position feedback.

VM1: The channel is set to virtual motor mode. The motor and encoder are simulated internally. The motor enable relay is held in the off state, but all other commands operate normally, including the position reference and snapshot. The channel is not able to read encoder signals in this mode.

8.3 Move commands

Basic commands for motor start and stop with use of a normal trapezoid or sinuid movement profile.

MA±nn Move to absolute position ±nn

Range: ±4.000.000

Version **2.004** new range for n: ±2.000.000.000

The motor moves to the absolute position given in the command. It follows a trapezoidal or a sinusoidal velocity profile (graph of velocity against time). With bit 2 of the "**CW**" command can choose between a trapezoidal or a sinusoidal profile. The motor accelerates from rest at the acceleration, set by the "**SA**" command (see page 39), until it reaches the velocity, set by the "**SV**" command (see page 38). At the end of the move, the motor decelerates at the rate from the "**DC**" command (see page 40) to stop at the desired final position. The position is entered in encoder counts.

It's possible to stop mapping (command XM) with command "**MA**". With bit 6 of command "**MW**" (see page 71) can select if the actual velocity or the velocity set with "**SV**" is selected to move to the target position nn. If bit 2 of "**ZW**" is not set, the current bound will not left. If necessary it comes to a stop and the move to the position is made in the other direction. If bit 2 of "**ZW**" (see page 50) is set the direction is dedicated with bit 3. If the current direction is not correct it will change.

Enhancement as of **version 1.008a**:
While clutching out of a map with "**MA**", the actual velocity is compared with "**SS**". If the velocity of the slave is lower resp. equal "**SS**", than "**SV**" is used.

If bit 1 of "**ZW**" is set to 1 the shortest way is done.

Enhancement as of **version 2.004**:

If the axis is running and it is made a "**MA**" or "**MR**" command and the breaking distance ist higher the the target position until now one more bound was driven.

If Bit 1 of "**ZW**" is not set now it is breaking down immediately and moving back to the wished position. If bit 1 of "**ZW**" is set, one more bound is moved.

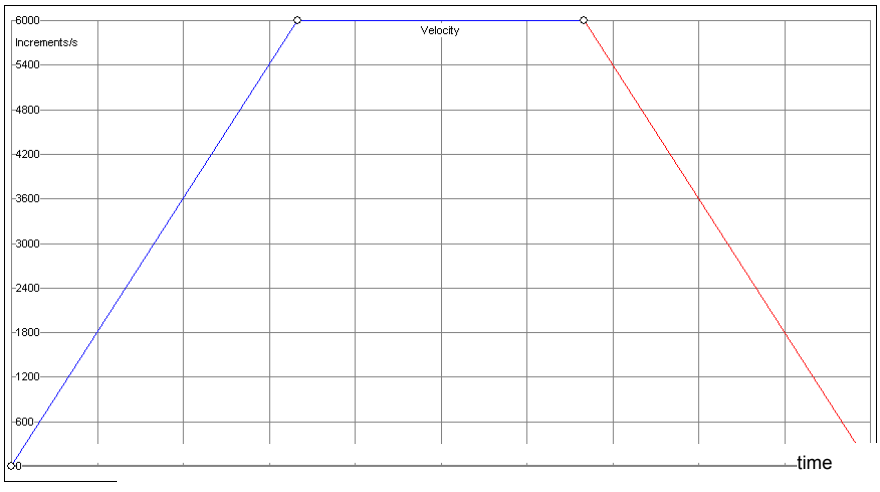


Figure 1: Trapezoidal profile

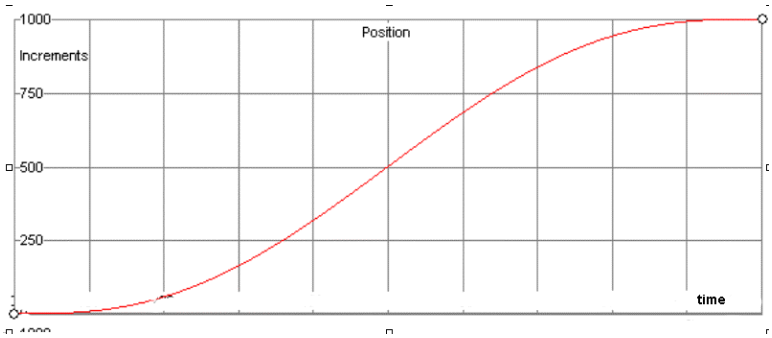


Figure 2: Position profile with trapezoidal velocity profile

If the move distance is small, the velocity is high, or the acceleration is low, the motor may not reach the set velocity within given move distance. In this case the motor follows a triangular velocity profile instead of a trapezoidal one.

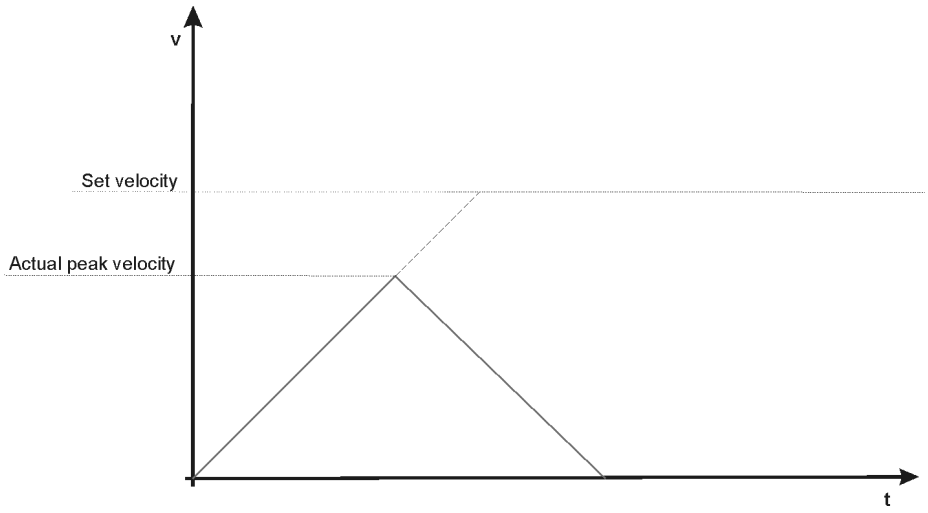


Figure 3: Triangular move profile

When PMCprimo executes an absolute or relative move command, it gives the '**M**' move prompt character. The move commands may only be used from the idle position control state.

Example 1:

PMCprimo User	Comment
`0.1> MA2000<CR>`	The motor moves to the absolute
`0.1M`	position + 2000 units
`0.1>`	The motor has reached the position 2000

Example 2:

PMCprimo User	Comment
`0.1> MA5000;WT256<CR>`	PMCprimo starts the "**WT**" command after the actual position
`0.1M`	of the channel reaches the target position (limited by
`0.1W`	the "**SW**" command see page 43).
`0.1>`	

The target move position is also checked against the current values of the user-defined position limits, set by the "**LH**" and "**LL**" commands. Is the target position out of the user-defined position limits, then the move command will not executed and PMCprimo displays an error message.

MR±nn Move ± nn units relative to current position

Range: ±8.389.000 encoder counts

Version **2.004** new range for n: ±2.000.000.000

PMCprimo performs a move similar to the absolute move above, but the move distance is defined relative to the current demand position. The move distance is entered in encoder counts. The channel follows a trapezoidal or a sinusoidal velocity profile (picture 01, graph of velocity against time).

The motor accelerates from rest at the acceleration, set by the "**SA**" command, until it reaches the velocity, set by the "**SV**" command. At the end of the move, the motor decelerates at the rate from the "**DC**" command to stop at the desired final position.

When PMCprimo executes an absolute or relative move command, it gives the '**M**' move prompt character. The move commands may only be used from the idle position control state. The relative movement is independent of the bits 1 – 3 of the "**MW**" command.

Enhancement as of **version 2.004**:

If the axis is running and it is made a "**MA**" or "**MR**" command and the breaking distance ist higher the the target position until now one more bound was driven.

If Bit 1 of "**ZW**" is not set now it is breaking down immediately and moving back to the wished position. If bit 1 of "**ZW**" is set, one more bound is moved.

Example:

PMCprimo User	Comment

`0.1> MR-3000<CR>`
`0.1M`
`0.1>`
	The motor moves from its current position
	3000 encoder counts in the negative direction

With the "**SV**" (or "**SS**") command the velocity could be set during executing a "**MR**" command to a new value. The change between "**SV**" and "**SS**" with the "**VJ**" command is possible every time, but the value from "**SV**" is necessarily higher than the value from "**SS**".

The target move position is also checked against the current values of the user-defined position limits, set by the "**LH**" and "**LL**" commands. If the move would take the motor outside the set position limits, then the 'target position outside limits' error message is again returned, and the move is not executed. The velocity during a move isn't changed if the velocity is set by a variable and the value of the variable is changed.

It is possible to enter mapping (see "**XM**" command page 74) with the "**MR**" command.

ST±nn Stop

Range: ±4.000.000

Version **2.004** new range for n: ±2.000.000.000

Without any parameter the motor stops under controlled deceleration, set by the "**DC**" command (see page 32). The "**ST**" stop command may be used during any motion to decelerate the motor to a stop. When the motor is stopping, PMCprimo gives the '**S**' stopping prompt character.

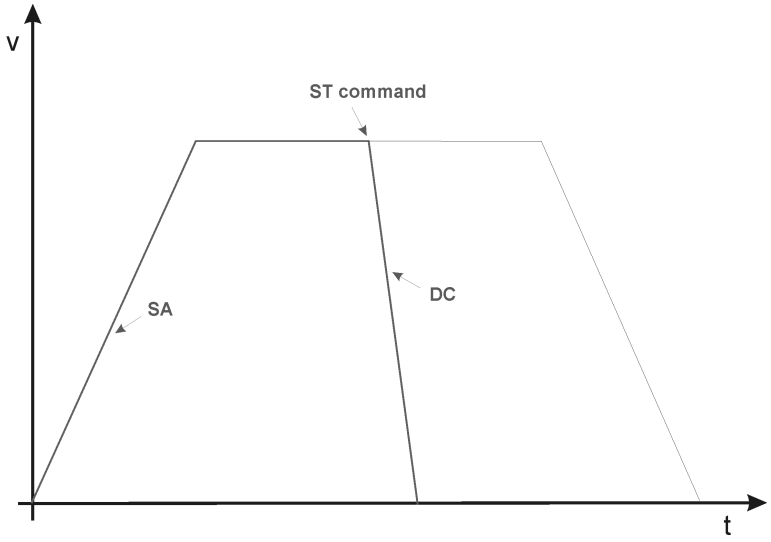


Figure 4: Move with normal stop

Without parameter nn the "**ST**" ramp is done immediately

With parameter nn it is made a move to the target position nn. It is possible to do it by every motion. During this time PMCprimo gives the "C"-prompt. This command can be used for clutching out of mapping to a defined position.

Enhancement as of **version 2.004**:

In the past the command stop to position (example "**ST0**") with mapping the active map was only kept until the deceleration was started. Then a linear ramp (ramp depending from "**CT**" or "**CL**") was used to stop the slave. If the master was stopped in that time then the slave still moved to the target position because the master slave link was already opened. The slave moved therefore sometimes more than the master.

Now with "**MW**" bit 6 it is possible to maintain the master slave link also while decelerating until the final target position is reached. This means that the slave is stopping as the master and then waiting until the master is moving again until the slave reaches the position. The setting of "**CL**" is always used. This means the bit 5 of "**MW**" (CT/CL setting) (see page 81) is ignored because with command "**CT**" it is not possible to reach the target position.

The "**ST**" command is executed at the current channel. If it is necessary to stop all channels then the "**GS**" (see page 63) global stop command should be used. With the command "**AX**" (see page 51) a sequence can abort without stop the motor.

AB Abort, emergency stop

The motor stops under controlled deceleration, set by the "**XA**" command (see page 40). The abort command may be used during any motion to decelerate the motor to a stop.

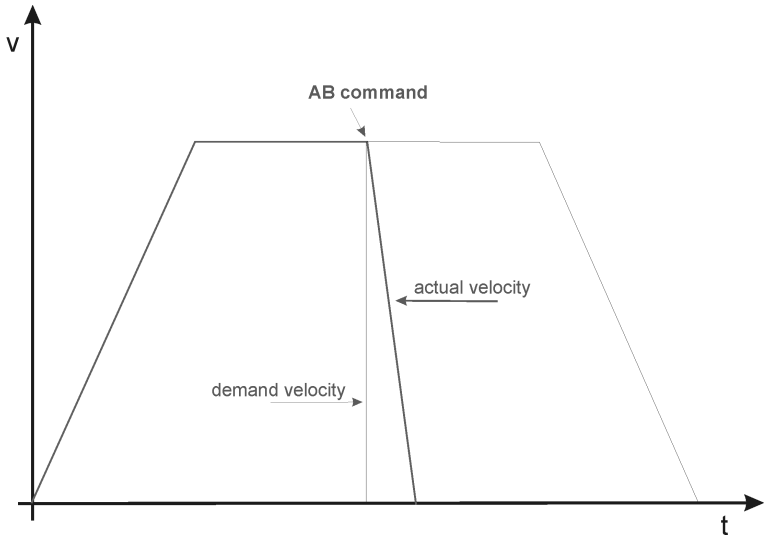


Figure 5: Move with abort

The "**AB**" command is executed at the current channel. If it is necessary to abort all channels, then the "**GA**" global abort command (seed page 63) should be used.

VC[±] Move at constant velocity

This command is used to move the motor at a constant velocity in the direction specified, without any target position. If the direction is not specified, the motor moves in the positive direction. PMCprimo accelerates the motor at the defined acceleration until it reaches the velocity set by the SV command (see page 38). If then controls the motor at constant velocity, until it is told to stop. While in constant velocity mode, PMCprimo gives the '**V**' velocity control prompt character.

Velocity control mode can only be entered from position control mode, and not directly from the motor off state.

Example:

PMCprimo User	Comment

`0.1> SA1000;SV2000;VC+<CR>`
`0.1V`
`0.1>`
	This command line set the acceleration to 1000 incr./sec.2, the velocity to 2000 incr./sec. and accelerates channel 1 at the defined acceleration until it reaches the velocity set by "**SV**" and moves with constant speed in positive direction.

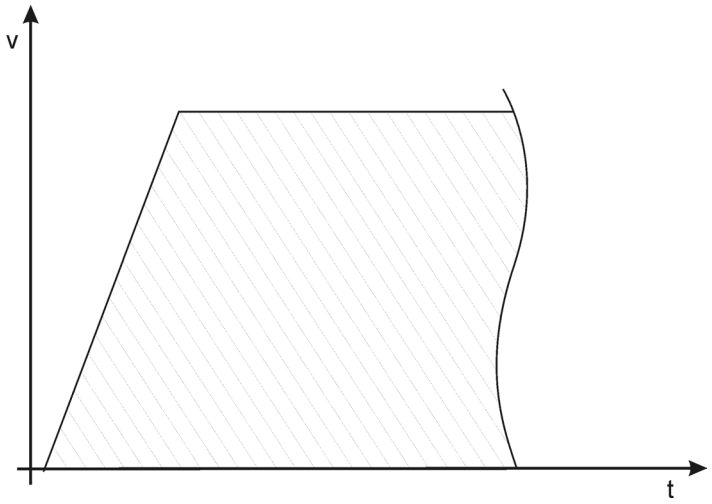


Figure 6: Constant velocity move

The velocity values "**SV**" and "**SS**" can also change during "**VC**".

IN[±] Initialise position

PMCprimo performs the initialisation sequence to find a zero position reference signal. In normal PC state the system gives the 'I' initialise prompt character while executing the initialisation sequence.

The motor accelerates to the set velocity in the specified direction. If no direction is specified, the motor moves in the positive direction. When PMCprimo detects a reference input signal, it resets the position counters to RF (see page 112) immediately. The motor then decelerates to a stop and moves back to the new zero position, if bit 3 of "**RW**" is not set.

This command may also be used in states other than "**PC**" when PMCprimo simply waits for a reference signal and sets the zero position accordingly.

If no reference input or marker input is defined, then the "**IN**" command returns an error message.

The "**IN**" command works independently of the settings of all the other reference commands. This is so that whatever the reference setup for normal running, the "**IN**" command always works normally. The exception to this is bit 3 of the "**RW**" reference options word, which disables the move back to the new zero point after the reference input is detected. The reference offset value is also effective during the initialisation, such that the position at which the reference signal is detected if defined as the absolute position given by the value of "**RF**", not necessarily zero. The "**RT**" command (see page 80) is also working. For more details please read the Reference Commands section later in this manual.

If the initialisation is abort due an error the sequence which has start the initialisation is stopped.

Enhancement as of **version 1.007**:
The bits 2 and 3 of ZW (page 50) are considered. Bit 2 defines if the direction for moving back to the reference signal is considered. Bit 3 defines the direction for the movement to the reference signal.

Example:

PMCprimo User	Comment
`0.1> IN+<CR>` `0.1I`	The motor moves in the positive direction until a valid reference input is seen. The motor stops, and moves back to the new zero position.

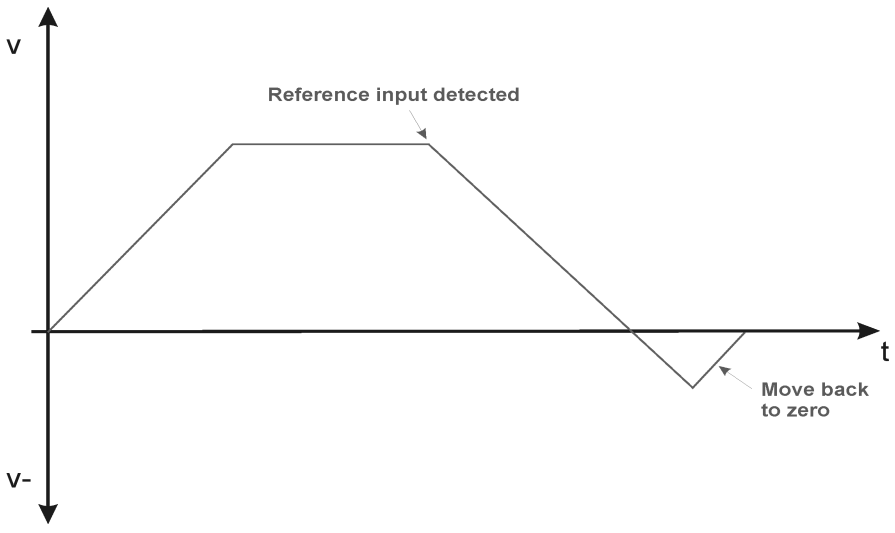


Figure 7: Initialisation to zero position

IB[±] Initialize position and bounds

This command is similar to the "**IN**" command but also sets the position bounds in addition to finding the zero position. In normal PC state, PMCprimo gives the 'I' initialise prompt character while executing the initialisation command. The motor accelerates to the set velocity (SV) in the direction specified. If no direction is specified, the motor moves in positive direction. When PMCprimo detects a reference input signal, it resets the position counters to zero immediately. The motor continues to move until a second reference input signal is detected and sets the position bounds to the distance moved since the first reference signal was received. The motor then decelerates to a stop and moves back to the new zero position.

This command may also be used in states other than "**PC**" when PMCprimo simply measures the distance between two successive reference signals and sets the zero position and bounds accordingly.

The "**IB**" command works independently of the settings of all the other reference commands. This is so that whatever the reference setup for normal running, the "**IN**" command always works normally. The exceptions to this are bit 3 of the "**RW**" reference options word, which disables the move back to the new zero point after the reference input is detected. The reference offset value is also effective during the initialisation, such that the position at which the reference signal is detected is defined as the absolute position given by the value of "**RF**", not necessarily zero. The "**RT**" command (see page 80) is also working. For more details please read the reference commands section later in this manual.

If the initialisation is abort due an error the sequence witch has start the initialisation is stopped.

With "**RW**" command (see page 110) you can avoid a move back to new zero.

Enhancement as of **version 2.000**:
It is possible to assign "**IB**" a variable e.g.: $VAR = IB. An "**IB**" is made and the result is set to $VAR. The value of "**SB**" is not changed.

PMCprimo User	Comment
`0.1> IN+<CR>` `0.1>`	The motor moves in positive direction until 2 reference signals are detected and moves back to the new zero position.

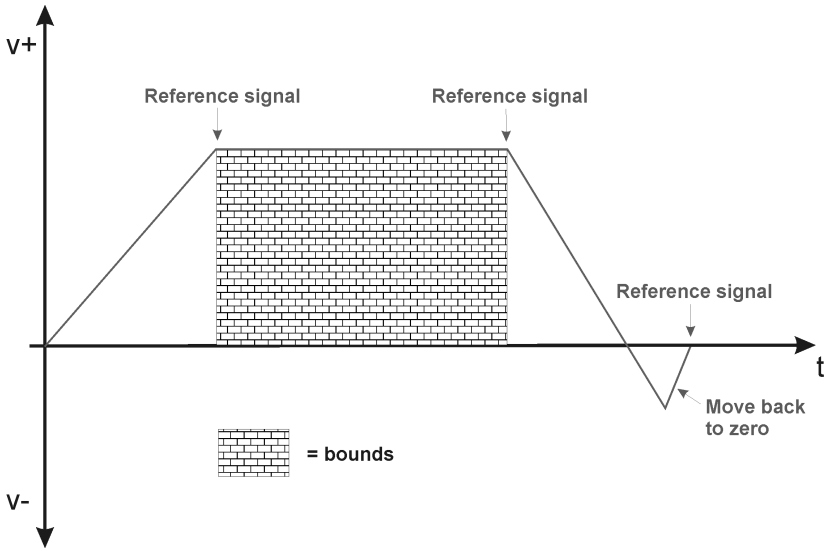


Figure 8: Initialise position and bounds

ID Initialize demand signal offset

Command as of **version 1.008.**

Under normal conditions, there may be some constant offset in the demand signal analogue output amplifiers which causes the motor to settle at a position slightly different to the required position. The “**ID**” command sets the system up to correct for this (assumed constant) offset in all subsequent position control operations. It must be used every time the system is powered on, when the system is in the position control mode, to set the actual position as close as possible to the required position. This is particularly necessary when the final position outside the final position window, and at the end of a move command it returns the error message .failed to reach target position. The “**ID**” command actually controlling the position, and it has no effect if the motor is not driving the system. Note that friction in the mechanical system can also cause a position offset after a move command is executed.

8.4 Parameter commands

Input and output of axis parameters

SVnn Set velocity

Range nn: 0 to 4.000.000
As of Version **2.004** new range for n: 0 to 40.000.000.

Default: 20000

This command is used to set the velocity of a channel in encoder counts per second. It may be used at any time including when the motor is already moving. The diagram below shows a typical velocity profile where the velocity is increased part way through a normal move (Figure 9).

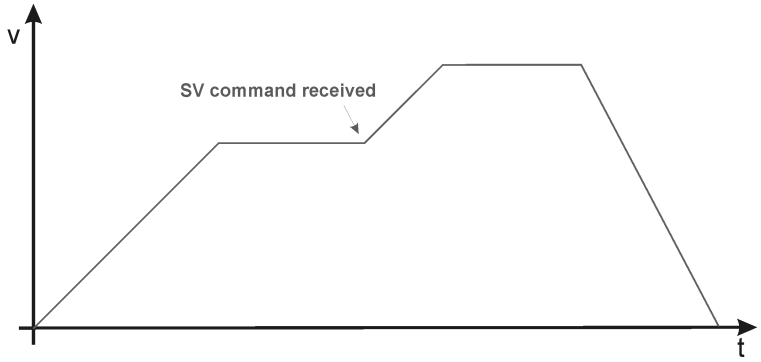


Figure 9: Move with change of velocity

Example:

PMCprimo User	Comment
`0.1> SV5000<CR>` `0.1>`	This sets the velocity to 5000 increments/second.

SAnn Set acceleration

Range nn: 1.000 to 2.000.000.000
Default: 100.000

This command is used to set the acceleration value in increments/second². The acceleration value can be changed every time, also during movement (beside during the execution of position tasks with “**MA**” or “**MR**” [page 31]). The lowest acceleration value is 1000.

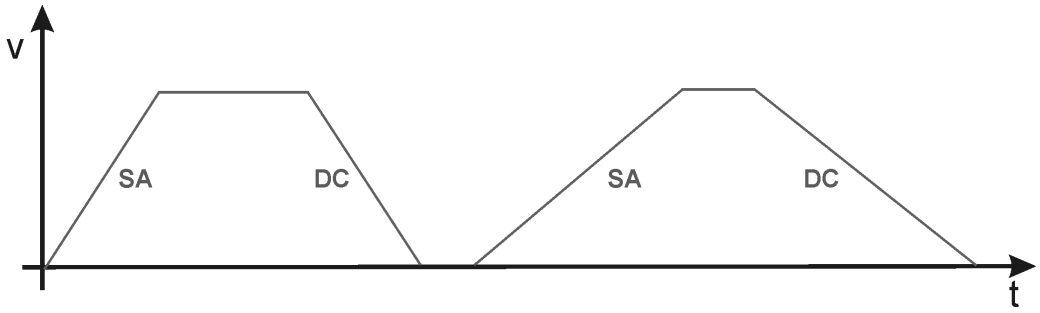


Figure 10: High and low acceleration (SA = DC)

Example:

PMCprimo User	Comment
`0.1> SA10000<CR>` `0.1>`	The acceleration is set to 10000 counts/ second^2.

DCnn Set deceleration

Range: 1.000 to 2.000.000.000
Default: 100 000

This command sets the deceleration of the channel to the specified value, in user units per second squared. The deceleration value is used in the move functions "**ST**", "**MA**", "**MR**", "**IN**" and "**IB**" for the deceleration ramp. The minimum deceleration value is 1000 counts/second^2. For the "**AB**" command the deceleration value is given by the "**XA**" command (see page 40).

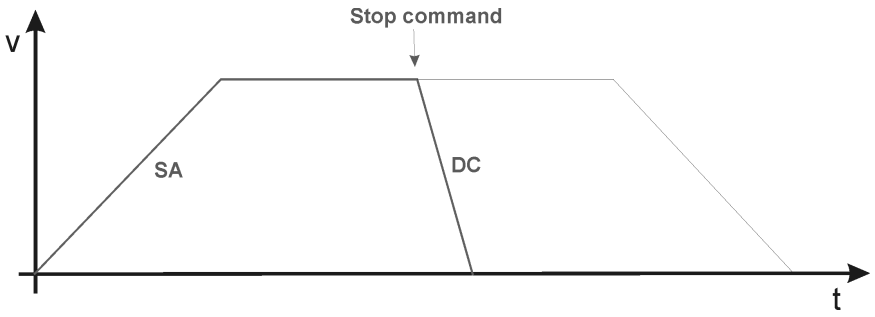


Figure 11: Effect of DC Command

Example:

PMCprimo User	Comment
`0.1> SA10000<CR>`	The acceleration is set to 10.000 counts/second^2 and the
`0.1> DC25000<CR>`	deceleration is set to 25.000 counts/second^2 on channel 1.

XAnn Set deceleration for AB command

Range nn: 1000 to 2.000.000.000
Default: 100 000

This command sets the deceleration for the "**AB**" command of the channel to the specified value, in user units per second squared. The deceleration value is used for the "**AB**" command (see page 33). The minimum deceleration value is 1000 counts/second^2.

SCnn Set creep distance

Range: 0 to 65.535
Default: 0

The normal trapezoidal velocity profile for a position move can be modified, to include a slow speed creep to the final required position. The creep distance is the distance from the final position over which the motor moves a the slow speed, set by the "**SS**" command. This may be used to minimise overshoot at high speed and accelerations. This command is only effective when "**VJ**" (see page 43) is set to zero.

Example:

PMCprimo User	Comment
`0.1> SC200<CR>` `0.1>`	This command sets the creep distance to 200 units. A position move command will now start to decelerate earlier than normal, such that the motor reaches the slow speed at least 200 units before the final required position.

SSnn Set slow speed

Range: 0 to 4.000.000
As of version **2.004** new range for nn: 0 to 40.000.000.

Default: 0

This command allows the user to set the speed of the slow creep to the final position, if required. The command is also used to set the slow speed to be used in slow velocity mode when "**VJ**" (see page 80) is set to 1. It is specified in counts/second. The value of "**SS**" has to be smaller than "**SV**" (see page 38).

The set velocity of "**SS**" is used for an adjustment move to correct a reference error although the actual velocity of the channel is equal to zero. "**SS**" is also used for adjustment velocity (change of "**MB**", "**MF**", "**SM**") if the velocity of the master channel is equal to zero.

Example:

PMCprimo User	Comment
`0.1> SS100<CR>`	Slow speed is set 100 counts/second.
`0.1>`	

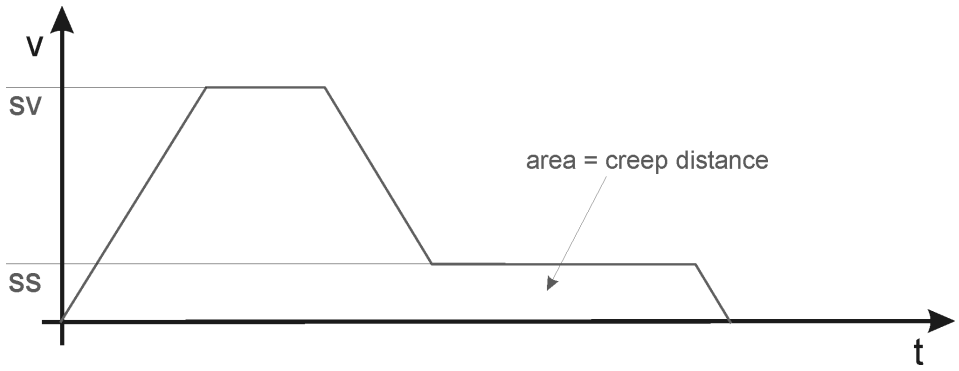


Figure 12: Move with creep to final position

VJn Set slow velocity mode

Range: 0 und 1
Default: 0

Setting “**VJ**” to 1 enables slow velocity mode. In this mode all moves are made at slow velocity as set by the “**SS**” command and the creep distance set by the “**SC**” command (see page 41) has no effect. Setting “**VJ**” to zero the axis changes to normal velocity mode where moves are made at normal velocity as set by the “**SV**” command (see page 38).

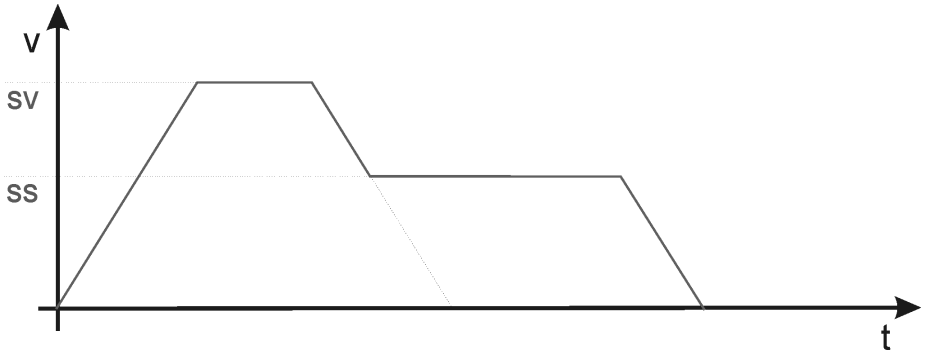


Figure 13: Normal/slow velocity mode

When switching between “**SV**” and “**SS**” during the execution of “**MA**” or “**MR**”, the predefined value for “**SS**” must be less than the value for “**SV**”, because otherwise the brake ramp cannot calculated correctly (brake ramp arduous).

SWnn Set window

Range: 0 to 65.535
As of version **2.004** new range for nn: 0 to 4.000.000

Default: 100

This command sets a window or tolerance around the required final position of a move. PMCprimo defines a move as finished when the demand position has reached the target position and the measured position is inside the window. It returns from the move state to the position control state only when the motor is within this window.

The set window is only used at move commands like “**MA**” (page 28) and “**MR**” (page 31).

Example:

PMCprimo User	Comment
`0.1> SW25<CR>` `0.1>`	This command sets the window to 25 counts. Thus PMCprimo returns the normal prompt at the end of a move only when the motor is within 25 counts of the required position.

PMCprimo reports “position is reached”, when the current position is within the window.

TInn Set timeout for window

(Set timeout for window)

Range nn: 0 bis 65.535
Default: 0

The check of “**SW**” is made after the time “**TI**”. The commands “**MA**” and “**MR**” are after the time “**TI**” ready.

Example:

PMCprimo User	Comment
`0.1> TI200<CR>`	Timeout for window is set to 200.
`0.1> MR200; MR200`	Two moves of 200 increments are made. Figure 14 explains the time “**TI**” between the two moves.

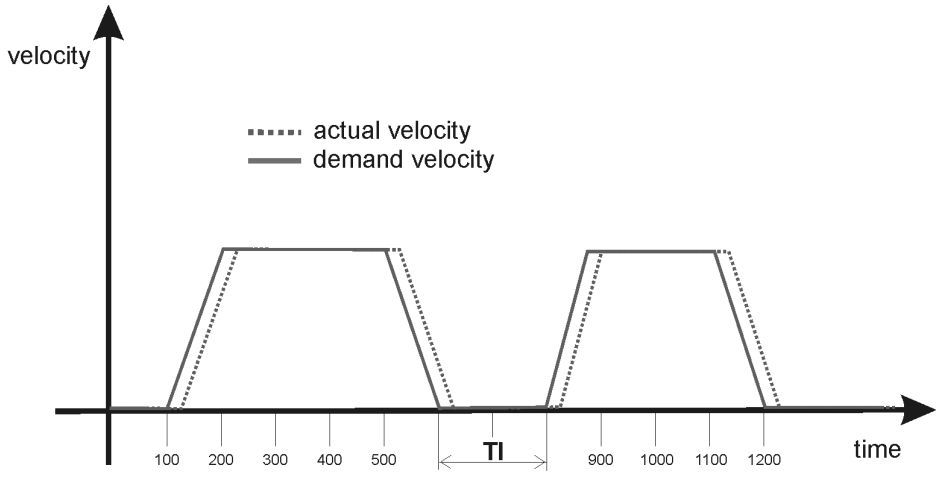


Figure 14: Timeout for window TI

BLnn Set backlash compensation distance

Range: 0 to 65.535
Default: 0

This command sets up a backlash compensation facility. It applies only to the "**MA**" and "**MR**" point-to-point move commands. It defines an extra distance that the motor moves each time it reverses direction, thus taking up any slack of backlash between the motor and the final output. It is defined in encoder counts.

Example:

PMCprimo User	Comment
`0.1> BL20<CR>` `0.1>`	This sets the backlash compensation distance to 20 counts. Each time the motor changes direction on successive move commands, the first move in the new direction is extended by 20 counts to take up the backlash.

ISn Set increment select code

Range: 0 to 8
Default: 0

This command selects the parameter which is incremented by the IP command (see page 47). Each channel has a separate "**IS**" value. The parameter selected is defined by the code as follows.

n	Parameter	Limits	as of version 2.004
0	no function		
1	current running speed	see below	
2	reserved		
3	Map base (MB)	±4000000	±2.000.000.000
4	Map offset (MF)	±4000000	±2.000.000.000
5	reserved		
6	Set velocity (SV)		
7	Scale map (SMn/m)	0 - 65535 for n	
8	Set bound (SB)	0 – 4000000	0-2.000.000.000

IS1: The lower limit for the current speed is 0. If the axis is running in normal velocity mode, the upper limit is twice "**SV**". If the axis is running in slow velocity mode, the upper limit is "**SV**". The increment affects only the current running speed and not "**SV**" or "**SS**" and is only effective until the motor stops. Subsequent moves start with a speed of "**SV**" of "**SS**" as appropriate.

IS3/IS4: Changes to map base or offset affect "**MB**/**MF**" parameters permanently. IF map base is changed, the change is subject to "**AV**" (see page 79) and the map base value wraps round at the master axis bound. If map offset is changed, the change is subject to "**AV**" and the map offset value wraps round at the slave axis bound.

IS6: The lower limit for the current speed is 0. If the axis is running in normal velocity mode, the upper limit is twice "**SV**". If the axis is running in slow velocity mode, the upper limit is "**SV**". The increment affects "**SS**" running with slow velocity mode and affects "**SV**" running with normal velocity mode. Subsequent moves start with the new value "**SV**" or "**SS**" as appropriate.

IS7: The lower limit for n of scale map (see page 69) is 0, the upper limit is 65.535.

IS8: Enhancement as of **version 1.008:**
The lower limit of n for the "**SB**" is 0, the upper limit is 4000000.

Example:

PMCprimo User	Comment
`0.1> SV1000<CR>`	Move velocity is 1000 counts/second
`0.1> IS1<CR>`	Select running speed for increment
`0.1> VC+`	Start motor running
`0.1V IP100;DV<CR>`	Increment running speed
`DV1100`	Running speed is now 1100
`0.1V IP100;DV`	Increment running speed
`DV1200`	Running speed is now 1200

IP±nn Increment selected parameter

Range: dependent on selected code

This command adds the value given to the parameter selected by "**IS**". This allows a selected parameter, such as motor speed, to be increased or decreased in steps by repeating a single command. The increment value would exceed its allowed range, it is set to its maximum or minimum value as appropriate. When a parameter value is changed with the "**IP**" command, then the new value is retained until the unit is turned off, or the previously saved values are restored. The changed values are not automatically saved, but may be saved with the "**SP**" command if required.

PTnn Encoder filter

Range: 0 to 10.000

It is possible to average the encoder signals over a defined space of time. Noisy encoder signals can be made to smooth signals. This command can be only used in status motor off ("**MO**").

If there is a torque control maximaum on a master axis, so t this fault can be seen also on the slave axis. With this filter the effect can be reduced.

With "**PT**" changes of velocity are registered at a later time. It also comes to a static offset for the averaged position.

CWbb Set control word

Range: 0 to 1111 1111
Default: 0001 0000

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command allows the user to write a value into the control word for the current channel. Note that the leading zeros may be omitted. The control word allows the state in which the axis module powers up to be defined, and allows the sense of the encoder input and of the command signal output to be reversed. The control word bit functions are described below (Bit 0 is right and Bit 7 left).

NOTE: The encoder and command signal sense should only be changed while the module is in the motor off state, as PMCprimo may be made completely unstable by reversing either of these. This facility is intended to be used only when initially connecting the module to the motor, to avoid having to rewire the system if the encoder connections are reversed. It also allows the logical positive and negative directions to be reversed under software control, by toggling both the encoder and output reversal bits in the control word.

Bit 0: Abortion if motor error:

0: Sequences waiting for a movement will stop.

1: sequences waiting for a movement will not stop.

Bit 1: Storing of positions in a battery-backed SRAM. This function is in PMCprimo 2+2 and 16+ possible.

0: No position is hold in the SRAM

1: Actual position is hold in two register of the SRAM. This bit can't be saved. If the position from the register should be read from SRAM after power on, the bit must be set in the startup sequence (see page 61). The position data will be copied in every tick of the system to two register of the SRAM. The position counter will be set to the value of the SRAM registers after power up if both registers are equal. If the two registers are not equal, an error will be reported on the specified channel.
Attention: This bit may only set, when all commands affecting the position are set (command “**FS**”, “**NB**”, “**MS**”, “**SB**”). If the bit is set too early, then the position in the ram may be wrong.

Enhancement as of version **2.004**:
If a motor is moved to a negative position and then PMCprimo is switched off and on then the position is calculated to a negative position. In former versions the position was calculated to a positive position. It is important that the motor is not moved more than the half bound length manually when the system is switched off.

Bit 2: Defines the profile of acceleration and deceleration

0: Trapezoidal profile of velocity for the commands “**MA**”, “**MR**”, “**VC**” and “**ST**”

1: Sinusoidal profile of velocity for the commands “**MA**”, “**MR**”, “**VC**” and “**ST**”

Bit 3: Behaviour in case of motor error:

0: Immediately motor off (command "**MO**")

1: Motor makes an emergency stop (command "**AB**") and after it motor off.

Bit 4: This bit defines the sense of the main analogue output for the motor command signal.

0: The command signal sense is normal; if the encoder is moved in the positive direction, a negative output voltage is produced at the command output.

1: The sense of the command signal output is reversed; if the encoder is moved in the positive direction, the command signal goes positive.

Bit 5: This bit defines the logical sense of the encoder input.

0: The encoder direction sense is normal; if encoder signal track A leads track B the motor is positive.

1: The encoder direction is reversed; if track A leads track B the motor is negative.

Bit 6: Command "**TO**" for encoder control

0: Command "**TO**" defines a check time for encoder control.

1: Command "**TO**" defines a way for encoder control.

Bit 7: This bit modifies the integral control action to help avoid the problem of wind-up during a move.

0: The integral term is active continuously. This is the normal setting.

1: The operation of the integral action is modified such that the position error is only added to the current integral total when the motor is static, in the idle position control state.

Example for command "**CW**"; bit 0:

PMCprimo User	Comment
`0.1: CW1<CR>`	Sequence waiting for movement should not abort. Because Bit 4 is set not to 1 the command makes a motor error.
`0.1: PC;MR10000;WT5000;SV<CR>`	The command "**MR**" makes a motor error
`0.1>`	status message "**PC**"
`0.1M`	status message "**MR**"
`0.1 Motor position error (01:48:28)`	
`0.1M`	
`0.1:`	
`0.1 SV=20000`	Despite the fact that a motor error occurs the sequence will continue and after 5 seconds "**SV**" is displayed.

ZWbb Set position control word

Range: 0 to 1111 1111
Default: 0100 0000

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command allows the user to write a value into the position control word for the current channel. The position control word allows the user to modify the operation of move commands in various ways. The position control word bit functions are described below (Bit 0 on right side, and Bit 7 on the left side).

Bit 0: reserved.

Bit 1: This bit controls whether "**SB**" has effects to an alignment move or to a move absolute or not.

0: The value of "**SB**" (see page 108) has no effect to move commands.

1: The value of "**SB**" is important for move commands ("**MA**"). The motor moves to the nearest position within bounds appropriate to the target position. (Example: ZC0;SB4000;MA6000 -> motor moves only 2000 counts).

Bit 2: This bit is used if an alignment move or a normal move must be constrained only in one direction.

0: The direction of a move is not specified by this bit.

1: The direction of a move is specified by Bit 3 of the position control word.

Bit 3: This bit defines the direction of a move if bit 2 is set to 1.

0: The direction of a move is positive.

1: The direction of a move is negative.

Bit 4: Enhancement as of **version 2.000**:
If bit 1 of "**ZW**" is set and the target position is a multiple of the value of "**SB**" for a move absolute ("**MA**") Bit 4 decides to move one bound or not.

0: If bit 1 is set and driveway=n*SB: no movement.

1: If bit 1 is set and driveway=n * SB: one position bound is moved.

Bit 5: As of **version 2.007**:
This bit controls whether "**SB**" has effects for the command "**WA**".
Dieses Bit entscheidet, ob die Zykluslänge"**SB**" für den Befehl "**WA**" berücksichtigt werden soll.

0: The value of **SB** has no effects for the command **WA**.

1: The value of **SB** has an effect for the command **WA** in the follow way:

Example:
SB10000

WA12000	It is wait for position 2000 or –8000 (12000%10000 = 2000)
WA-12000	It is wait for position 8000 or –2000 (-12000%10000 + 10000 = 8000)

The WA command is finished within one bound.
It is not possible that WA wait forever if for example WA-1000 was started but the motor moves in positive direction if the bit is set to 1.

Bit 6: reserved

Bit 7: Enhancement as of **version 2.000**
If the axis is not moving in mapping an automatic bound correction is possible with this command. Therefore the bound set by "**SB**" is multiplied with the scale map "**SM**". With an odd gear transmission a reference sensor is no longer necessary.
Example: SB4096; SM1,3
The bound for it is 1365,33. Therefore a drift of one increment every 3 bound would happened. With correction the bound set two cycles to 1365 and one cycle to 1366. The calculated bound can be shown with the command "**GM**".

0: Automatic correction is not active
1: Automatic correction is active

8.5 Sequence commands

This section describes the sequence commands. They provide comprehensive facilities for defining, reviewing and executing complex command sequences. Command execution on different channels may be explicitly specified to proceed in sequence or in parallel, with each channel working independently.

A sequence consists of one or more lines of commands which can be executed by issuing the appropriate "**XS**" command. Unlike command strings, it is possible to execute one sequence while another is already in progress provided the motor state allows it.

On PMCprimo units, all sequences are stored by the host processor system, and downloaded to the appropriate axis as required. A sequence which consists only of commands for one motor is downloaded complete to the current motor channel when it is first called up, and is executed by the axis module itself. The sequence is first analysed by the host, and split into its various components to be executed on each motor, and on the host. Any parts of the sequence that can be executed on a single channel are downloaded to that channel as a single-channel sequence.

The local single-channel sequences are assigned sequence numbers by the host system automatically, and are not available to the user. If a sequence is modified or deleted, the host system modifies or removes the appropriate local sequences as required.

This sequence analysis and downloading it to the different channels is completely automatic and requires no interaction of the user.

Commands and sequences may execute in sequence or in parallel on different channels. The default case is for all commands to be executed in sequence, expect where specified explicitly by the "**CP**" change channels in parallel command.

Example 1 (first channel 1 moves to 20000, then channel 2 moves to 10000):

PMCprimo User	Comment
`0.1> ES PRG1<CR>`	Enter sequence PRG1
`002E CH1;MA20000<CR>`	Channel 1 move to 20000
`003E CH2;MA10000<CR>`	Channel 2 move to 10000
`004E NS<CR>`	End of PRG1
`0.1>`	

Example 2 (channel 1 and channel 2 moves simultaneous):

PMCprimo User	Comment
`0.1> ES PRG1<CR>`	Enter sequence PRG1
`002E CP<CR>`	Executing simultaneous ON
`003E CH1;MA20000<CR>`	Channel 1 move to 20000
`004E CH2;MA10000<CR>`	Channel 2 move to 10000
`005E CE<CR>`	Executing simultaneous OFF
`006E NS<CR>`	End of PRG1
`0.1>`	

The move commands on both channels are started at the same time, if sequence PRG1 is executed.

The channels in the example above are not linked together. If more general synchronisation is required, there are commands available to manipulate links between the motor channels.

If some degree of synchronisation is required between operations taking place in parallel, then this may be specified by defining the sequence as a set of parallel sequence components which themselves execute in sequence. An example is a set of point-to-point moves in two axes on an XY table. Each two axis move consists of two moves running in parallel on two motors, but both moves must be complete on both motors before the next action can take place. The next example shows this, and also demonstrates how sequences may be nested.

Example:

PMCprimo User		Comment
0.1>	ES PRG1<CR>	
002E	CP<CR>	Executing simultaneous ON
003E	CH1;MA20000<CR>	Sequence PRG1: Channel 1 and channel 2
004E	CH2;MA10000<CR>	starts move absolute at the same time
005E	CE<CR>	Executing simultaneous OFF
006E	NS<CR>	
0.1>		
0.1>	ES PRG2<CR>	
002E	CP<CR>	Executing simultaneous ON
003E	CH1;MA0<CR>	Sequences PRG1: Channel 1 channel 2
004E	CH2;MA0<CR>	starts move absolute at the same time
005E	CE<CR>	Executing simultaneous OFF
006E	NS<CR>	
0.1>		
0.1>	ES PRG3<CR>	
002E	XS PRG1;XS PRG2<CR>	Sequence PRG3: Executing sequence PRG1
003E	NS<CR>	and if PRG1 is finished, it executes sequence PRG2
0.1>		

ES[Name] Enter sequence

The sequence commands allow the user easily to build up complex sequences of machine operations and store them in PMCprimo. A stored sequence may be called up and executed with a single command.

The ES command is used to enter command sequences into PMCprimo. PMCprimo responds with a “002E“ prompt for the sequence entries. Each entry in a sequence can be any valid command line. Command strings on one command line are accepted as one sequence entry. Sequences entries may also include commands to execute other sequences, to allow sequences to be nested. To end the sequence, use the “**NS**” command, and PMCprimo then returns to normal operation. The sequences can be nested up to 30.

Every line of a sequences includes one command, one command string or a comment started with **#**. The “**NS**” command exits the enter sequence command.

The name of a sequence starts always with a letter. The length of the sequence name is limited up to 20 signs. The only sign except letters and numbers is the underscore-sign ‘_’.

Example 1::

PMCprimo User	Comment
`0.1> ES PRG10<CR>`	Enter sequence PRG10
`002E IN-<CR>`	Initialise channel in negative direction
`003E RP3;MR100<CR>`	Move relative 3 times 100 counts in positive direction;
`004E MA2000<CR>`	Move to absolute 2000
`005E NS<CR>`	Exit enter sequence
`0.1>`	

Example 2 (delete a sequence):

PMCprimo User	Comment
`0.1> ES PRG10<CR>`	Enter sequence PRG10
`002E NS<CR>`	Exit enter sequence
`0.1>`	

NS End of sequence

This command is the command for PMCprimo for the end of a sequence. The last command in every sequence is the “**NS**” command.

LOOP Loop for sequence

The name '**LOOP**' is reserved for a cyclic sequence. A sequence named “**LOOP**” is running cyclic with low priority. After power up the “**LOOP**” starts not automatically. The command “**XP**” (see page 57) starts the program cyclic. With the command “**XS**” (see page 57) the sequence is running one time. The command “**GX**” (see page 63) or “**GX**” “**LOOP**” stops the loop. The command “**ZT**” (see page 25) shows if “**LOOP**” is active.

The following commands are not allowed in a “**LOOP**”-Programm:

“**SP**“: Save data (page 22)

"**CP**“: Change to paralell mode (page 62)

"**DM**“: Continuous display mode (page 163)

"**TR**“: Enable trace mode (page 164)

“**CD**“: Change the operating mode (page 16)

„**XM**“: Execute map (page 74)

“**XS**”: Execute sequence (page 57)

“**SK**“: Software key (page 26)

“**LM**“: Display position table (page 73)

“**XX**“: Execute map virtual (page 75)

“**LB**“: List bus variable value (page 186)

“**LV**“: List variable value (page 186)

LS[Name] List sequence

This command allows the user to examine a sequence that has previously been entered into the system. The sequence is listed on the terminal. If no sequence name is given in the command, PMCprimo lists the names of all sequences which are currently defined. It is also shown, if the sequence is a host- or a node-sequence. In a node-sequence are only commands of a specific node. If the sequence is defined as an input-function, it can run without the host and therefore much faster.

It is not possible to make a change at the sequence with this command.

Example 1: List sequence PRG10

PMCprimo User	Comment
`0.1> LS PRG10<CR>`	List sequence PRG10

`ES PRG10`
`IN-`
`RP3;MR100`
`MA2000`
`NS`	

`0.1>`	

Example 2: List all sequences

PMCprimo User	Comment
`0.1> LS<CR>`	List all sequences
`TEST (NODE 0)`	sequence TEST is node-sequence
`PRG10 (HOST)`	sequence PRG10 is host-sequence
`PRG15 (NODE 1)`	sequence PRG15 node-sequence
`0.1>`	

XSName Execute sequence

This command tells PMCprimo to execute sequence Name. The normal status message for each part of the sequence are printed on the display as they are executed. The sequence aborts automatically if any error occurs. The "**GX**" (see page 63) command aborts executing sequences. It is not possible to execute the same sequence a second time if it is in use. If the sequence is triggered before it is finished PMCprimo gives the error message „Cannot execute sequence while still busy".

Example: Execute sequence PRG10

PMCprimo User	Comment
`0.1> XS PRG10<CR>`	execute sequence PRG10
`0.1M`	PMCprimo executes sequence PRG10
`0.1>`	sequence PRG10 finished

XPName Execute sequence parallel

This command starts a sequence parallel. If a sequence is started from another sequence with "**XP**" the sequence runs parallel. The normal status message for each part of the sequence are printed on the display as they are executed. The sequence aborts automatically if any error occurs. The "**GX**" (see page 63) command aborts executing sequences. It is not possible to execute the same sequence more than one time. If the sequence is triggered before it is finished PMCprimo gives the error message „Cannot execute sequence while still busy".

Note: You may only start 25 sequences parallel.

Example: Parallel start of a sequence

PMCprimo User	Comment
`0.1: ES PRG11`	Enter sequence PRG11
`002E XP PRG12`	Executing PRG12 parallel.
`003E SV`	Display velocity
`004E NS`	Exit enter sequence
`0.1: ES PRG12`	Enter sequence PRG12
`002E WT10000`	Wait 10 seconds
`003E SA`	Display acceleration
`004E NS`	Exit enter sequence
`0.1: XS PRG11`	Execute sequence PRG11
`0.1:`	
`0.1 SV=20000`	The velocity is displayed immediate.
`0.1 SA=100000`	After 10 seconds the acceleration is displayed.
`0.1:`	

RP[nn] Repeat command line

Range: 1 to 4.000.000, or no parameter

This command tells PMCprimo to repeat the commands on the current command line nn times. If no repeat count is given, PMCprimo repeats infinitely. Only one “**RP**” command is allowed on any command line. The “**ER**” end repeat command may be used to break out of the loop at the end of the current loop. The “**AX**” or “**GX**” command breaks out of the loop immediately.

Example:

PMCprimo User	Comment
`0.1> RP5;MA100;MA0<CR>`	Channel 1 moves to 100 and back to 0
`0.1M`	The command line will be executed 5 times
`0.1>`	

ER End repeat

This command allows the user to exit from a repeat loop cleanly, at the end of the current loop. The “**ER**” command is independent from the given number of repeat counts (for example look “**IF**” command page 59).

IFexpression Execute commands if expression is true

The IF command causes the commands following it to be executed only if the expression is true (i.e. non-zero). This command affects all commands following it up to the end of line or the next "**EL**" command, whichever comes first. The expression following the "**IF**" command may consist of a simple variable, a comparison or a combination of expressions. A comparison uses the operators ==, !=, <, >, <=, >=. Expressions can be combined using the operators &&, ||.

Example 1:

PMCprimo User	Comment
`0.1>  RP;IF($X<10);$X=($X+1);SV($X*10000);WT1000;EL;ER`	
`0.1>`	This command line repeatedly increments "**$X**", sets a new speed and waits 1 second until "**$X**" becomes equal to 10.

Example 2:

PMCprimo User	Comment
`0.1>  IF($V1==2&&($X<=20));XSPRG3;EL;XSPRG21<CR>`	
`0.1>`	This example shows a more complex expression in the "**IF**" command.

EL Else - execute commands if expression is false

The "**EL**" command causes the commands following it to be executed if the preceding "**IF**" or "**II**" command (page 125) evaluated to false, or zero. The "**EL**" command must either appear on the same line as the associated "**IF**" or "**II**" command, or as the first command on the following line. If the "**EL**" command appears in any other position, all commands following it up to the end of line are simply ignored.

Example:

PMCprimo User	Comment
`0.1>  IF(($B!=0)&&($A>=100));XS PRG_A;EL;XS PROG99`	
`0.1>`	This command line executes sequence PRG_A if "**$B**" is non-zero and "**$A**" is greater than or equal to 100. Otherwise sequence "**PROG99**" is executed.

IXName Execute commands if program is active

The commands after “**IX**” are only done, if the sequence named 'Name' is active. It comes to an error-message if the sequence doesn't exist. The command “**EL**” (see page 59) can also used.

AX Abort command execution

This command aborts execution of any command strings or sequences running on the current channel but without stopping the motor.

If “**AX**” is issued while the channel is in mapping or constant velocity states, the motor is left in mapping or constant velocity as appropriate. If “**AX**” is issued while the channel is executing a move or waiting, the move or wait is finished normally but following commands are aborted.

PMCprimo User	Comment
`0.1>  VC+<CR>`	Motor 1 is in constant velocity mode
`0.1V  RP;DP;WT200<CR>`	Display actual position every 200ms
`0.1V DP 1000`	
`0.1V DP 1500`	
`0.1V DP 2000`	
`0.1V DP 30000  AX<CR>`	Abort command
`0.1V`	Motor 1 is still in constant velocity mode

BK Break out of sequence

This command causes the system to break out of the current sequence and to continue executing commands in the calling sequence (if any). This is useful for terminating a sequence early depending on the state of an input line.

PMCprimo User	Comment
`0.1>  LS PRG10<CR>`	List sequence PRG10
`ES PRG10`	
`CH0.1;II1-;BK`	break out of sequence if input 1:1 is low
`CH0.1;MA1000`	channel 1 move to 1000
`NS`	end of sequence
`0.1>`	

ASName Set autostart sequence

Default: 0

This command is used to set up a command sequence to execute automatically when PMCprimo starts up, after all the saved setup parameters and configuration details are loaded from the flash memory. If no sequence name is given, PMCprimo prints the current autostart sequence name. If 0 is given to AS, no autostart sequence is defined.

CH[t]n Change channel

range for t: 0 to 59 (maximum 60 Network nodes)
range for n: 1 to 3 for PMCprimo Drive
1 to 4 for PMCprimo 2+2
1 to 20 adjustable with command "**CD**" (PMCprimo 16+)
1 to 10 for PMCprimo Drive2

This command allows the user to switch between motor channels. It may be used at any time. If no parameter is given for t, PMCprimo uses automatically node number 0.

Note for PMCprimo Drive: Channel 1 is the controlled channel, channel 2 is only an encoder or a PMCtendo DD4 as a real channel, channel 3 is only a virtual channel (or a real encoder).

Example:

PMCprimo User	Comment
`0.1>  CH1.1;MA200;CH2.1;MA100<CR>`	Channel 1 from node 1 moves to 200 and afterwards Channel 1 from node 2 moves to position 100.

CP Change to parallel mode

This command allows the user to switch between motor channels in a sequence, and allow simultaneous command execution on more than one channel. The command CP is only allowed in sequences.

Example:

PMCprimo User	Comment
`0.1>  ES PROG1<CR>`	Enter PROG1
`002E  CP<CR>`	simultaneous ON
`003E  CH0.1;MA200<CR>`	channel 1 from node 0 move to 200
`004E  CH1.1;MA100<CR>`	channel 1 from node 1 move to 100
`005E  CE<CR>`	simultaneous OFF
`006E  NS`	end of sequence
`1.1>`	

The move commands in sequence PROG1 are started at the same time if sequence PROG1 is called.

CE End of parallel mode

This command tells PMCprimo the end of a simultaneous execution of commands. The command CE is allowed in sequences. If there is no CP command before the CE command PMCprimo gives the error message „Still in parallel mode“.

GS Global stop

This command sends a stop command ("**ST**" see page 32) to all channels. The "**GS**" command is a global stop command for all channels.

GA Global abort

All channels stop with their "**AX**" deceleration ramp. The "**GA**" command sends an "**AB**" command (see page 33) to all channels.

GF Global motor off

All channels go in the motor off state immediately. The "**GF**" command sends a motor off command (see page 27) to all channels.

GE Global wait off

This command stops all running wait commands. The "**GE**" command is a global "**WE**" command (see page 92).

GX[Name] Global abort execution

This command sends an "**AX**" command (see page 60) to all channels and stops all running sequences. Starts a sequence with an "**GX**" command, all other sequences are stopped and than all commands following the "**GX**" will be executed. If a sequence name is given, only the specified sequence will be stopped.

MEName Set motor off error sequence

Range: sequence name or 0
Default: 0

This command sets up a sequence to execute when any motor off error occurs on the current channel. If no sequence name is given, PMCprimo prints the current motor error sequence name for the current channel. To disable the motor error sequence on this channel, set "**ME**" to zero.

Follow errors executes a motor error sequence if defined with “**ME**”:

Code	Error
5	High position limit exceeded
6	Low position limit exceeded
23	Memory full
24	Motor position error
29	Motor timeout. No encoder signals
37	Reference error outside limits
38	Reference correction overrun
39	Reference timeout
56	Limit switch detected
74	Timout for position update
117	Analogue input low limit exceeded
118	Analogue input high limit exceeded
141	Mastervalue not in map
172	Drive error: heat sink temperatur to high
173	Drive error: overvoltage
174	Drive error: feedback (cable break, short circuit, short to ground)
175	Drive error: undervoltage
176	Drive error: motor temperatur
177	Drive error: internal aux. voltage not OK
178	Drive error: overspeed
179	Drive error: EEprom checksum error
180	Drive error: Flash-Eprom checksum error
181	Drive error: brake (cable break, short circuit, short to ground)
182	Drive error: motor phase missing
183	Drive error: internal temperature to high
184	Drive error: output stage not OK
185	Drive error: I2t max value exceeded
186	Drive error: supply power - BTB/RTO
187	Drive error: A/D-Converter defective
188	Drive error: regen circuit faulty
189	Drive error: supply phase missing
190	Drive error: system error (system software)
230	Buffer for storing reference position is full!
249	Drive error: kommutation error
250	Variable $%s[%d] is not defined

Code	Error
251	Node number not defined (command <%c%c)
252	Node number %d is already in use (command <%c%c)
253	No CAN-device found
254	Cannot execute drive command, because buffer overflow
255	Warning: SRam battery low
256	Drive error: Enable switched on before AS-option
257	Drive error: Earth short circuit
258	Drive warning from CAN
259	%d bytes Compact Flash memory free
260	Cannot execute <XM> while motor is running
261	Command QA0:CAN-Adr can only done on the host
262	The analog output <%ld> is not available! (command <%c%c>)
263	Reserved for Soft-SPS
264	Reserved for Soft-SPS
265	Reserved for Soft-SPS
266	Reserved for Soft-SPS
267	Reserved for Soft-SPS
268	Reserved for Soft-SPS
269	Reserved for Soft-SPS
270	Reserved for Soft-SPS
271	Reserved for Soft-SPS
272	Reserved for Soft-SPS
273	PD command not possible! PMCtendo DD4 version wrong(< 4.94)
274	Node %d, Drive command not completed !
276	Battery buffered memory is used by SoftPLC
277	The maximum possible position (SB*SM) is greater than the position limit of %ld (command <%c%c>)
278	The predefined map 'LINEAR' can't be changed

8.6 Map commands

This section describes the commands to use the position mapping or „Software Gearbox“ facilities.

The position mapping commands provide a mechanism for defining the required position of a slave motor with respect to a given master channel, for all possible positions of the master channel. The master channel may itself be controlling a motor, or it may simply be monitoring the position of, for example, a line shaft to which other motors must be synchronised. The mapping itself consists of a table of slave position offsets relative to the master position, with one entry for every possible master position.

When the map is executed on the slave axis, it’s demand position is calculated by using the current master position as an index into the map table, and then adding the offset value in the table to the master position.

In a simple case, this allows any linear gear ratio to be defined between the master and the slave axis. This linear gearbox is inside PMCprimo as map table „LINEAR“. In more complex applications, it allows the system to mimic the action of non-linear systems, where the position of the slave motor has some more complicated relationship to the master axis position. This can be used to replace eccentric gearbox mechanism, crankshaft linkages, cam operated pushrods, or almost any mechanical linkage or transmission system with equal ease, simply by defining the required map table.

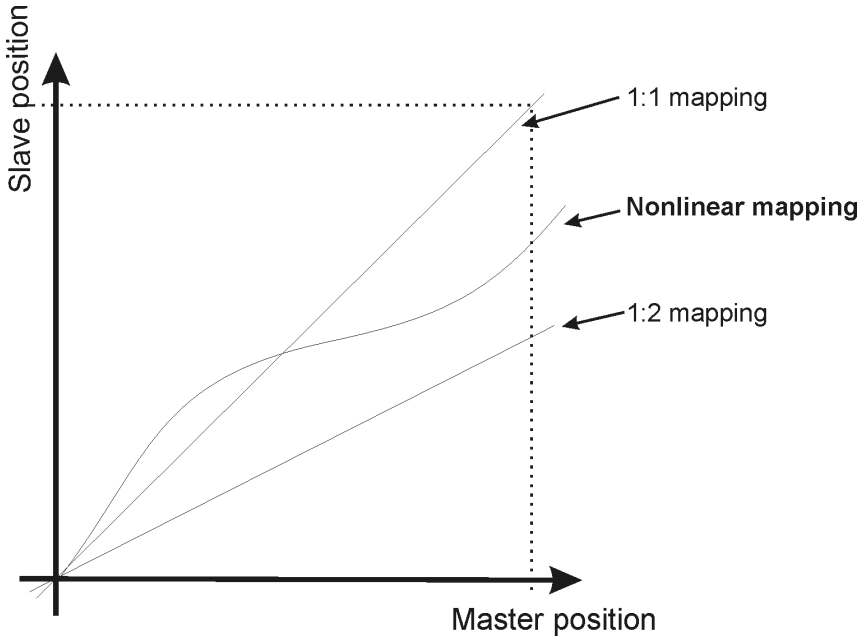


Figure 15: Simple position maps

On a machine where both master and slave axes are only moving from point to point within a finite range of positions, for example on a XY table, the mapping simply needs to be defined over the range of master position actually used, and the mapping itself then defines the slave position range. This is the simplest application. In such a system, there is no need to set up the position bounds on either channel. They may be left at the default value, since the motor positions are constrained within fixed ranges and the motors cannot move continuously in one direction.

The diagram below shows a typical position mapping, where the slave channel is following some position profile relative to the master axis. This could be, for example, the path for a cutting tool on a machine where the master axis speed varies to maintain a constant linear tool speed across the material. This could not be easily achieved by other methods, which would rely on keeping the cutter motion synchronised in time with the master axis motion. This could only be set up for one specific master axis speed or profile, and would need to be programmed to allow the machine to be run at a different speed. Using the position mapping mechanism, the master axis speed may be varied at any time, even during a cut, and the cutter stays on the correct path, as defined by the mapping.

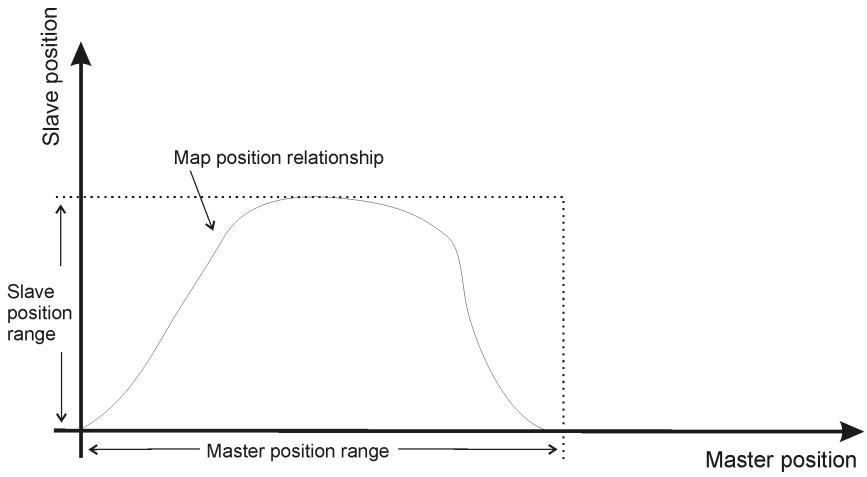


Figure 16: Position mapping over a defined range

The example of Figure 16 could also apply to a machine where the master axis is a continuous rotary axis, such as a line shaft, with the slave axis profile repeating once for every cycle of the master axis. In this case the master bound position must be set to the master axis cycle length, and also copied to the slave axis with the "**MP**" map bound command.

On a machine where either or both axes are cyclic, the position bounds on each axis are set to the cycle length for that axis. The mapping must be defined such that the required slave axis position is continuous at the points where the master and slave axes pass their respective bound positions. If this is not done, the discontinuity in the mapping at the position bounds could give rise to very sudden changes in speed and position of the slave when the bound positions are reached.

If the slave position is not continuous across the map boundary, the difference between the required slave cycle length and the actual cycle length given by the mapped master position bound appears as a small shift of the slave axis relative to the master axis. This error accumulates over several cycles of the machine and behaves as an apparent drift of the slave axis. This type of problem is difficult to spot, but can be prevented by careful definition of the position map table.

The mapping should be defined over the master position range from zero to at least the master axis bound position. In addition, the master axis bound position value must be known by the slave axis, so that when the master axis position wraps around to zero at the bound position, the slave axis knows how to adjust is mapped demand position in order to maintain is speed across the master wraparound point. This is done automatically by PMCprimo when the "**ML**" (see page 76) map link command is used or if the master bound is changed.

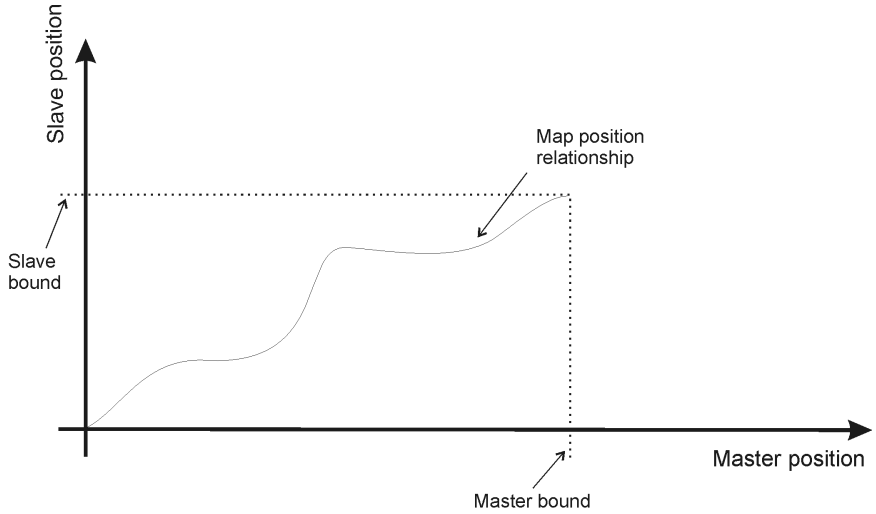


Figure 17: Position map for a cyclic machine

The diagram in Figure 17 shows a system where the master and slave axes both complete one cycle in the same time, although they cover different distances, and the master and slave bound positions are coincident. It is not necessary for the master and slave bound positions to coincide, or for the slave axis bound position to repeat in the same interval as the master axis bound. In fact, in any linear ratio mapping other than the simple 1:1 case, the bound positions will not coincide and will not repeat at the same rate. This does not cause any problems in executing the map. It is also possible for the required slave position to go outside the slave bound value without any problems. In this case, the slave channel compensates automatically it's wraparound when it passes the bound position.

The position mapping between the master and slave channels may be modified by the map base and map offset commands. These appear to be similar, but have subtly different effects in practice. They shift the position map relationship along either the master position or the slave position axis. The map base value is subtracted from the master axis position before indexing into the map table. This has the effect of shifting the map curve to the right on the graph. The map offset value is added to the slave position value calculated in the mapping. This has the effect of moving the map curve up the graph. These two parameters allow any or all slave axes to shifted or rotated relative to the master axis, even while executing a mapping. An example of a similar situation is the ignition timing on a car engine; the timing adjustment involves rotating the distributor shaft relative to the crankshaft, so that the spark is generated earlier or later in the engine cycle.

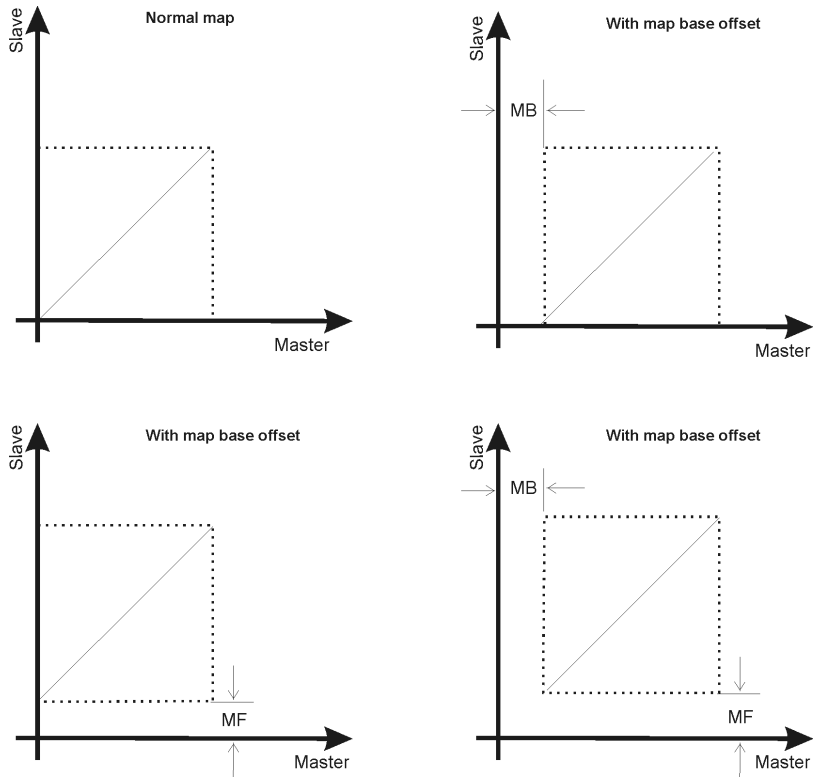


Figure 18: Effects of map base and map offset

Data for the maps are entered as absolute slave axis positions. Each map table entry represents the required slave motor position at the given master axis position.

Maps are stored on the host. This allows a map common to more than one channel to be defined only once, thus saving memory space.

The map data are manually downloaded to the current channel by using the "**TM**" (see page 76) transfer map command. This may be executed as part of the autostart sequence to avoid any delay the first time a map is executed.

Only the given map entrees are saved in PMCprimo. PMCprimo interpolates linearly between the position intervals to generate the full resolution map.

The Software Differential allows a slave motor to be synchronized so that it follows either the sum or the difference between the positions of two master axes. This is analogous to a differential gearbox. The slave axis must be linked to a master axis with the "**ML**" command (see page 76), and to a second master axis (called the differential axis) with the "**NL**" command (see page 76). "**LW**" bits 4- 6 (see page 83) control how the slave axis uses the position information from the two master axes, and "**LW**" bit 0 controls whether the master or differential axis transmits its demand or actual position to slave axes.

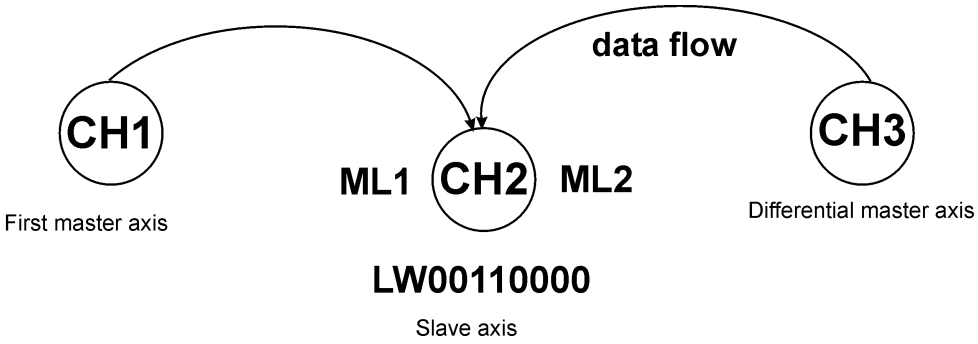


Figure 19: Example setup for software differential

Note that the master channels may be in any state, provided the map links are set up as required. Also note that it is important to set the "**LW**" options (on master and slave axes) before executing the "**ML**" map link command. Any change to "**LW**" (error message when "**ML**" is active) is not applied until the slave channel is unlinked and re-linked to the master axis!

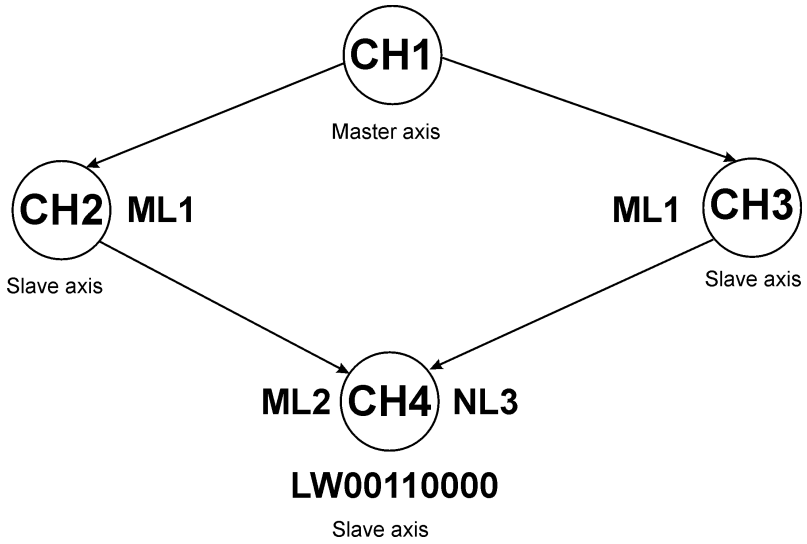


Figure 20: Example setup of software differential (complex)

The various parameters governing position mapping can be summarised in the following equations. Unless noted all parameters apply to the slave axis.

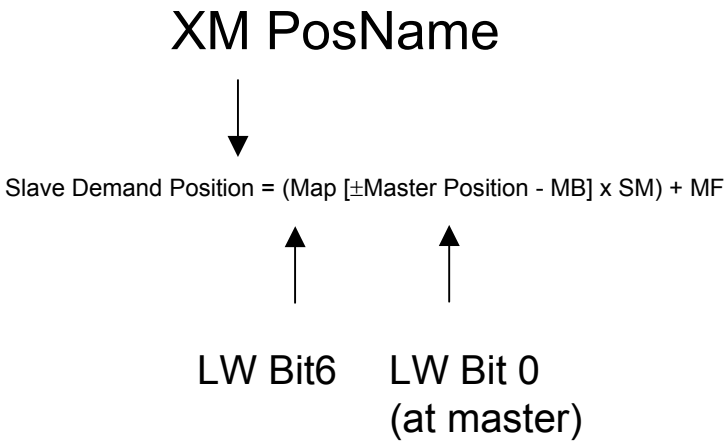


Figure 21: Position mapping as an equation

$$S = \{Map\ [\ \pm(M_p\ \pm(M_d \times SB_{Master}/SB_{Diffmaster})) - MB] \times SM\} + MF$$

LW Bit5

S	Slave Demand Position
M_p	Primary Master Position (ML)
M_d	Differential Master Position (NL)
SB_{Master}	Set Bound Master
$SB_{Diffmaster}$	Set Bound Differential Master

Figure 22: Position mapping as an equation (differential)

EMName Enter map

This command is used to enter position mapping tables into PMCprimo. Each entry in the table is the (signed) absolute position on the slave axis at the current master channel position given by the table entry number.

It is possible to enter 200 maps.

Example 1:

PMCprimo User	Comment
`0.1: EM TESTMAP<CR>`	Name of position map
`Masterbound: 1000<CR>`	set bound master
`Number of values: 3<CR>`	number of table entries (here 3)
`001M 0<CR>`	first table entry 1
`002M 500<CR>`	
`003M 1000<CR>`	
`004M NM<CR>`	End of enter map
`0.1:`	

Example 2: Delete a map table

PMCprimo User	Comment
`0.1>  EM Testmap<CR>`	Name of position map
`0.1>  Masterbound: NM<CR>`	End of enter map

NM End of enter map

This command is the last entry in a map table. If there is no entry before “**NM**” in the table, the map itself is deleted.

LM[Name, n] List map

Range n: 0 to 10000 without 1

This command allows the user to examine map data previously entered into PMCprimo. With n >1 PMCprimo lists n values. The listed map can not be changed with this command. If there is no name given, PMCprimo lists the names of all maps which are inside the node.

Example:

PMCprimo User	Comment
`0.1>  LM Testmap<CR>`	list map "Testmap"
`EM TESTMAP`	name of the map
`1000`	masterbound
`3`	number of table entries (here 3)
`0`	first table entry (always 0)
`500`	
`1000`	
`NM`	End of enter map
`0.1>`	

XM[Name] Execute map

This command puts the currently selected axis into the position mapping mode, where it's demand position is calculated according to the specified mapping from another (master) channel's position. The master channel position is transferred to all slave channels; this is set up automatically when the slave channel is linked to the master channel with the "**ML**" map link command (see page 76). While a channel is executing a position mapping, it gives the '**X**' map mode prompt character. The stop or abort commands are used to exit from map mode.

The linear position map is inside PMCprimo, named "LINEAR".

The map "LINEAR" is used for the special case of linear mapping. If map "LINEAR" is executed, the slave channel executes a scale of "**SM**" mapping, and it is not necessary to transfer the map with "**TM**", and it is not possible to enter map dates for the map 'LINEAR' into PMCprimo.

If there is no name given, PMCprimo displays the current active map.

The map could be executed if the master is stopped or stationary.

1. Master axis is stationary:

   The slave moves to its demand position given from the map table . If the software clutch is enabled (bit 1 of "**MW**" command page 81), the slave is waiting until the master comes to the right position and clutches in. If the velocity is less than 500 incr/sec the slave is clutching in without software clutch. For this the master velocity should be averaged with the "**BT**" command (set "**BT**" on the slave channel) (see page 173).

2. Master axis is moving:

   To execute a map if the master is moving, then the slave axis can wait for the required mapped position to pass the current slave position, and then start following the mapping as it passes. The software clutch (bit 1 of "**MW**" command) must be activated if the master is moving.

If the "**XM**" command is part of a string, the commands following "**XM**" are not executed until the slave axis has finished the alignment move or has synchronized with the master axis using the software clutch.

The action of the "**XM**" command is modified if tension control is on, as set by "**AM1**" command. The bits 4 and 5 of the "**AW**" command (see page 176) defines a special behaviour of "**XM**".

With speed mapping (bit 4 of command "**MW**" see page 81) only linear maps (XM LINEAR) are allowed.

Enhancement as of **version 2.000**
It is possible to set "**XM**" to a variable. In this case the actual mapnumber is set to the variable. If mapping is not active the variable is set to 0xffff.

XVName, nn **Execute map virtual**

Range: -6.000.000 to 6.000.000

With this command it is possible to get the actual slave position of the master position 'nn' in map 'Name'. It can read without “**XM**” a slavepostion for example in a variable. All settings (“**MB**”, “**MF**” etc) has the same effect like “**XM**” (see page 74).

The map must be transferred with “**TM**” to the axis. The map link must be executed on the axis.

Example:

PMCprimo User	Comment
`0.1> $VAR1=XV MAP_1,1000<CR>`	Get slave position if master position is 1000 and Map is MAP_1.
`0.1> $VAR1<CR>`	Get value of $VAR1
`0.1>`	
`0.1> $VAR1=2345 (0x00000929)`	Display value of $VAR1

XXName, nn **Length of alignment move**

With this command the necessary alignment move for the position table “name” can be aquired, which are made automatically after the start of the command “**XM** Name” (Bit 0 of “**MW**” [page 81] must be set to 0). Thus it it possible to make the alignment move before the execution of “**XM**”. The command “**XX**” must always run on the slave axis.

The position table must be assigned to the axis with the command “**TM**”. On the axis “**ML**” must be run.

Example:

PMCprimo User	Comment
`0.1> ml2`	Axis 0.1 should be slave and axis should be master
`0.1> zc0`	Slave position set to 0
`0.1>`	
`0.2> zc300`	Master position set to 100
`0.2> ch1`	
`0.1> $XX=xx linear`	Result of XX is saved in variable $XX
`0.1> $XX`	Value $XX
`0.1>`	
`0.1> $XX=300 (0x0000012C)`	Slave must drive 300 increments, to get to desired position

TM[Name] Transfer map data

This command tells PMCprimo to transfer the specified map to the current motor channel. The map table is stored in the flash of the Host and it is necessary to transfer the map to the axis one time before "**XM**". This could be necessary for example if the map generator has generated a new map.

If "**TM**" is not followed by the name of a map, PMCprimo lists all the existing maps of the actual Node.

ML[t.n] Map link slave axis to master axis

Range t: 0 to 59 (depends on number of installed nodes)
Range n: 1 to 3 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 20 adjustable with command CD (PMCprimo 16+)
1 to 10 (PMCprimo Drive2)

This command links the current channel to the specified master channel in preparation for execution of a position mapping. Note that a slave channel must be linked to some master channel before executing a map. n defines the master axis and t defines the node number there the channel is present.

Example:

PMCprimo User	Comment
`0.1> CH0.2<CR>`	Change to channel 2, node 0
`0.2> ML0.1<CR>`	Channel 1, node 0 is master of channel 2, node 0
`0.2> CH0.3<CR>`	Change to channel 3
`0.3> ML0.2<CR>`	Channel 2 is master of channel 3
`0.3>`	

NL[t.n] Map link to differential axis

Range t: 0 to 59 (depends on installed nodes)
Range n: 1 to 3 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 20 adjustable with command CD (PMCprimo 16+)
1 to 10 (PMCprimo Drive2)

This command links the current channel to the specified master differential channel in preparation for execution of a software differential position mapping. Note that a slave channel must be linked to some master channel before executing a map. n defines the master axis and t defines the node number there the channel is present. If the "**NL**" command is given with no parameter, the system prints the channel number of the differential master axis for the current channel, if it is linked. For differential see also "**LW**" on page 83.

UL Unlink slave axis from master axes

This command is used to unlink a slave axis from it's master axes after finishing a position mapping. Note that channels may be left linked to their master axis if required, while other commands are be executed. It is only necessary to use the "**UL**" command if the slave channel needs to be linked to a different master axis.

Example:

PMCprimo User	Comment
`0.1> CH3<CR>`	change channel 3
`0.3> UL<CR>`	unlink
`0.3> ML1<CR>`	channel 3 is linked to channel 1
`0.3>`	

MB±nn Set map base offset for master map positions

Range: ±4.000.000
As of version 2.004 new range for n: ±2.000.000.000

Default: 0

This command sets a map base value. This value is subtracted from the master axis position (resultant position after differential and "**SN**" scaling have been applied) before using the data for the map input. An alternative description is that it defines the base position of the mapped region on the master channel, such the slave channel is mapped into the master channel position for the range (MB) to (masterbound+MB). Normally the mapping is defined over the range from zero to the master axis bound. The "**MB**" parameter allows the slave channel to be advanced or related to the master axis. A change of "**MB**" during mapping involves a compensation with the velocity defined with "**AV**" (see page 79) and the alignment acceleration "**AA**" (see page 80).

MF±nn Set slave map position offset

Range: ±4.000.000
Default: 0

This command sets a map offset value. This value is added to the slave position obtained from the mapping. This allows the slave axis map profile to be rotated or shifted relative to the master axis. If the "**MF**" is changed during mapping, the alignment move is made with the velocity "**AV**" and the acceleration of "**AA**" (see page 80).

SMn,m Scale mapping

Range n: 0 to 65535
Range m: 1 to 65535
Default: 1,1

This command is used to set a scale factor for mapping. It is used on the slave channel, not on the master. The required absolute position on the slave channel, as defined by the mapping from the master position (or result of any software differential), is multiplied by the first parameter value and divided by the second value. This allows a wide range of scale factors to be realized, while keeping a simple integer ratio scale function. Note that changing the map scale factor while executing a map can give erratic results, because this may change the required slave axis demand position by a large amount. To avoid this problem "**AV**" and "**AA**" (see page 79) can be used to smooth out the changes in "**SM**".

Example: Scale map 1096/361

⇒ The demand position is calculated from the position from the map table multiplied with 1096/361.

PMCprimo User	Comment
`0.1> SM 1096,361<CR>`	The scale map is set to 1096/361. Channel 1 is slave
`0.1>`	

BR Set map scale factor from bounds ratio

This command is used to automatically calculate the map scale factor ("**SM**" see page 78) on the slave axis from the ratio of the slave bounds to the master bounds. The scale factor calculated is equivalent to SB'slave'/SB'master'. The "**BR**" command should only be executed after executing the "**ML**" command. The correct sequence of operations is shown in the following example.

Example:

PMCprimo User	Comment
`0.1>  SB4000<CR>`	Set bound 4000 channel 1
`0.1>  CH2<CR>`	Change channel 2
`0.2>  SB5000<CR>`	Set bound slave 5000
`0.2>  ML1<CR>`	Map link to channel 1
`0.2>  BR<CR>`	Calculate scale map
`0.2>  SM<CR>`	edit scale map
`SM=5000,4000`	

AVn[,x] Set map base/offset/scale factor adjustment

Range n: 0 to 200%
As of **version 2.006**: New parameter x: 0 to 6
Default: 0

This command sets the adjustment speed for any change in the map base, map offset (MB- MF command see page 77) or map scale factor ("**SM**" command see page 78) values entered while the axis is executing a map. The "**AV**" command is used to make large adjustments smoothly by spreading them over several time steps. If "**AV**" is set to zero, then any map base, offset or scale factor adjustment is performed immediately in one step.

If "**AV**" is not zero and the axis is executing a mapping, then the adjustment of map base or offset is limited to a set maximum speed, given by the sum of the adjustment velocity and the current (instantaneous) motor velocity. The adjustment velocity is a power of two fraction of the current motor speed, defined by the value of "**AV**". This means that the adjustment speed scales automatically with the machine speed, such that the value of "**AV**" may be chosen for correct operation at full machine speed without causing unnecessarily quick adjustments at lower machine speeds.

The parameter "**AV**" is given in per cent of the actual demand velocity. If the actual demand velocity is 0, the adjustment is done by the velocity of "**SS**". The parameter for "**AV**" must be set on the slave channel.

As of **version 2.006**:
The command "**AV**" was improved. With the additional parameter "x" the resolution could be increased. The calculation is x * (0.1% pow n).
The maximum value for n is 6
Example:
AV5,1 set the resolution to 0.5%
AV 5,2 set it to 0.05%
AV10,0 set it to 10%
AV10 is like earlier version 10%

AAn Set map base/offset/scale factor adjustment acceleration

Range: 0, 1000 to 2.000.000.000
Default: 0

This command sets the adjustment acceleration for any change in the map base, map offset or map scale factor values entered while the axis is executing a map. The "**AA**" command gives an acceleration ramp for the change between the demand velocity to the adjustment velocity. "**AA**" is given in counts/second squared and is given to the slave channel. If there is no value given to "**AA**", the acceleration is not limited. The slightest value for acceleration is 1000.

MWbb Set map options word

Range: 0 to 1111.1111
Default: 0

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command allows the user to modify the operation of position mapping in various ways (Bit 0 is right and Bit 7 left):

Bit 0: This bit controls the behaviour of the system when a position mapping is executed.

0: Software clutch disabled, the slave axis calculates its demand position as required by the mapping, and executes a normal trapezoidal move to align itself to that position before going into the mapped state.

1: Software clutch is enabled. This is used when it is required to lock a slave channel to a master axis which is already moving. In this case, the slave channel remains it's current position, until the calculated demand position from the mapping approaches the current slave position. The slave channel then ramps up to the required speed in such a way as to reach this speed at the correct mapped position. The time for this clutch acceleration ramp is defined by the "**CT**" clutch time command (see page) or if Bit 5 is set, the acceleration ramp is defined by the "**CL**" clutch length command. If bit 2 of "**MW**" is set as well as bit 0, then the "**XM**" command executes with an automatic offset adjustment. In this case, the map offset parameter "**MF**" is adjusted to maintain the current relative positions of the master and slave axes when the "**XM**" command is executed, and mapping starts immediately.

Enhancement as of **version 2.002**:

With speed mapping (bit 4 of command "**MW**" must set) it is possible now to go in mapping without software clutch. Therfore the velocity of master must be lower than 500 increments per second.

Bit 1: This bit defines if an alignment move (caused for example by "**MB**" or "**MF**") depends on set bound or not. Example: The position of the axis is –10000 increments and the set bound is 10000 increments. If bit 1 is set to 1 and MB is set to +20000 the alignment move is zero.

0: Set bound value has no effect for alignment move.

1: The target position for the alignment move is set to the nearest correct cyclic position within "**SB**".

Bit 2: If mapping starts with the alignment move, this bit specifies that it is constrained to move in only one direction.

0: The direction for an alignment move at the beginning of mapping is not defined.

1: The map alignment move at the beginning of mapping is always in one direction defined with bit 3 of "**MW**".

Bit 3: This bit sets the direction of the map alignment move, if it is constrained by setting bit 2 of the map options word to 1.

0: The map alignment move is in the positive direction.

1: The map alignment move is in the negative direction.

Bit 4: This bit is used to execute mapping as speed mapping, where the slave speed is related to the master speed by mapping, instead of relating the slave and master positions.

0: Position mapping

1: Speed mapping

With speed mapping follow must regard:
If the slave receives the master values via CAN-Bus the master speed is calculated about the master positions. A change of the master position made by referencing or for example the comand ZC is affected to the slave speed.
If the master and slave are on the same node (slave gets no master values via CAN-Bus) changes of positons of the master have no affects to the slave speed.

Enhancement as of **version 2.002**:

The master velocity can be averaged in speed mapping with the command "**BT**". Damping of rapid movements of the master is possible with it.

Enhancement as of **version 2.005**:

Now it is possible to interrupt speed mapping with stop to position. In this case bit 6 of command MW is ignored.

Enhancement as of **version 2.006**:

The alignment acceleration "**AA**" is now also with speed mapping active if the map ratio is changed with the "**SM**" command. With the old versions the speed ration was switched immediately without a ramp.

Bit 5: The acceleration ramp of the 'Software Clutch' facility is defined as clutch time or as a clutch length.

0: The acceleration ramp is defined with clutch time "**CT**" (page 85).

1: The acceleration ramp is defined with clutch length "**CL**" (page 86)

Enhancement as of **version 2.004**:

When "**ST**" is used the map is still active when decelerating. The setting of "**CL**" is always used. This means the bit 5 of "**MW**" (CT/CL setting) is ignored because with "**CT**" it is not possible to reach the target position.

Bit 6: It is possible clutching out with "**MA**" or "**MR**". Therefore the velocity can select with this bit.

0: Move with actual velocity.

1: Move with velocity set by "**SV**" (see page 38).

Enhancement as of **version 2.004**:

Clutching out with ST the map is done until stop if the bit is set. Only the clutch length CL is used, bit 5 of command MW has no relevance.

Bit 7: Enhancement as of **version 1.006a:**
Activate an automatic bound correction. The slave bound is set automatic to the slave position at the master bound and its scale map. The actual bound can be displayed with the command "**GM**" (page 162).
Enhancement as of **version 2.000**
If the axis is not moving in mapping an automatic bound correction is possible with this command. Therefore the bound set by "**SB**" is multiplied with the scale map "**SM**". With an odd gear transmission a reference sensor is no longer necessary.
Example: SB4096; SM1,3
The bound for it is 1365,33. Therefore a drift of one increment every 3 bound would happened. With correction the bound set two cycles to 1365 and one cycle to 1366

Special case "**MW 101**":

If bit 2 of "**MW**" is set as well as bit 0, then the "**XM**" command executes with an automatic offset adjustment. In this case, the map offset parameter "**MF**" is adjusted to maintain the current relative positions of the master and slave axes when the "**XM**" command is executed, and mapping starts immediately. The value of "**MF**" could be set by the user or by a sequence after the "**XM**" command is finished.

LWbb Set map link options word

Range: 0 to 1111.1111
Default: 0

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command allows the user to modify the operation of the map link function. It also enables the „Software Differential" function. The value is entered as a binary number, with each bit controlling a different aspect of the position mapping (Bit 0 is right and Bit 7 is left).

If the „Software Differential" is enabled, the demand position of the slave is calculated from the sum or the difference of both master positions. The master is defined with the "**ML**" command and the differential master is defined with the "**NL**" command (see page 76). The bit functions for the link options word are described below.

Example:

PMCprimo User	Comment
`0.1: LW<CR>`	set map link options word on slave channel
`0.1 LW=00000000`	before map link
`0.1: LW00110000<CR>`	enable software differential
`0.1: ML2<CR>`	channel 2 is master
`0.1: NL3<CR>`	channel 3 is differential master
`0.1: PC<CR>`	enter position control mode
`0.1> XM LINEAR<CR>`	execute linear map
`1X`	the demand position of the slave is calculated master position minus differential master position. There is no link between the masters necessary.

Bit 0: This bit controls what data from the master axis give the demand position of the slave. Normally the slave is set up to follow the master demand position, as this gives smoother motion on the slave axis.

0: Slave follows the demand position of the master.

1: Slave follows the actual position of the master.

Bit 1: This Bit defines if the correction of the reference error is made on the master or slave.

0: The correction of the reference error is made on the real master

1: The correction of the reference error is made on the slave.

Bit 2: Enhancement as of **version 2.000**
In speed mapping it is possible to restrict the slave velocity to a maximum (value of "**SV**") and a minimum (value of "**SS**"). The bits 2 and 3 of "**MW**" for definition of direction are active in this case.

0: Function speed limit is not active.

1: Function speed limit is active, if bit 4 of "**MW**" is set to 1

Bit 3: not used

Bit 4: This bit enables the „Software Differential". The demand position of the slave depends on the sum or the difference of two master axes.

0: Software differential disabled.

1: Software differential enabled.

Bit 5: This bit controls whether the sum or the difference of two master axes gives the demand position of the slave axis.

0: The master positions are added.

1: The master positions are subtracted.

Bit 6: This bit controls the sign of the calculated demand position.

0: The master position data is used unchanged.

1: The master position data is negated before being used by the current channel.

Bit 7: not used.

CTn Set clutch time

Range: 1 to 20.000
Default: 1

When the software clutch is enabled, a slave axis does not go immediately into the mapped state on performing an execute map command. Instead, it goes into a holding state and waits for the projected master position to pass the projected slave position at the end of the slave acceleration ramp, as given by the clutch ramp time factor. The clutch time defines the number of time steps in milliseconds, taken for the slave axis to accelerate from rest to the required mapped speed.

This command sets the acceleration ramp time for the „Software Clutch“ facility. The software clutch is enabled by bit 0 of “**MW**”, the map options word. The software clutch is defined as a time value with “**CT**” if bit 5 of the “**MW**” is set to 0. It allows a slave channel to be mapped onto a master axis that is already moving.

The maximum clutch time is n=20000 ms and the minimum clutch time is 1 ms. The value for the clutch time is given in milliseconds.

Example:

PMCprimo User	Comment
`0.1> CT16<CR>`	clutch time is set to 16ms
`0.1>`	

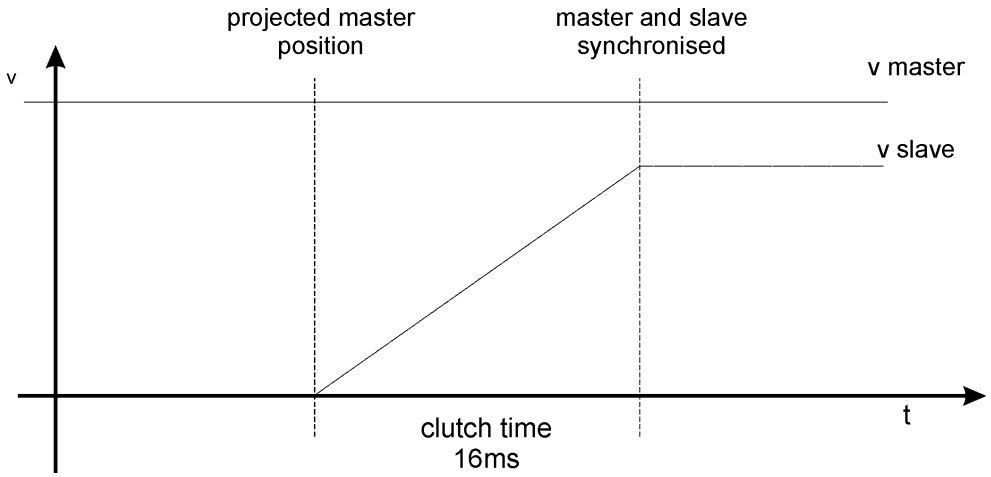


Figure 23: Software clutch

CLn Set clutch length

Range: 0 to 4194304
As of version **2.004** new range for n: 0 to 2.000.000.000

Default: 0

This command sets the acceleration ramp length for the „Software Clutch“ facility. The software clutch is enabled by bit 0 of the “**MW**” (see page 81), the map options word. The software clutch is defined as a length with “**CL**” if bit 5 of the “**MW**” is set to 1. It allows a slave channel to be mapped onto a master axis that is already moving.

The value of n = 4194304 is the maximum length and n=0 is the minimum length. The parameter is set on the slave but the given length depends on master counts.

Example:

PMCprimo User	Comment
`0.1> CL2000<CR>`	clutch length is 2000 master counts
`0.1>`	

CIn Set clutch window

Range: 0 to 65.535 increments

Enhancement as of **version 1.008**

This command defines a window for software clutch.

If CI ≠ 0 and the way for alignment is lower than “**CI**” the slave makes an alignment move. If CI ≠ 0 and the way for alignment is higher than “**CI**” the slave is using software clutch. Bit 1 of the MW command is ignored if CI ≠ 0. But bit 2 and 3 is used for the “**CI**” command. If the master is moving “**CI**” is ignored. The command “**CI**” can’t used in speed mapping (bit4 of “**MW**”) and in tension control.

MTn Masterposition timeout

Range: 4 to 10.000 milliseconds
From **version 2.006**: Range: 0 to 1000 milliseconds

In the PMCprimo network the master position is transferred all 4 milliseconds. During the 4 milliseconds the positions for the clock of 1 millisecond are interpolated. For monitoring the transfer of master positions with CANopen the command “**MT**” can be used. If no position is received in the defined time “**MT**” an error message is given out and the motor will be switched off.

Enhancement as of **version 2.002**:
It is possible to make an assignment to a variable.

8.7 Wait command

The wait commands are most useful in sequences. The wait command is a Host command. The number of wait commands, which are executed in parallel is not limited. The command "**WT**" is a host command. PMCprimo doesn't return a „**W**" status message to indicate that it is waiting on a specific channel.

WTnn Wait for time

Range: 0 to 65535

This command tells PMCprimo to wait for the given time, before proceeding to the next command. The wait time is specified in milliseconds.

Example: Waiting for 2 seconds

PMCprimo User	Comment

`0.1> MA2000;WT2000;MA0<CR>`
`0.1M`
`0.1M`
`0.1>`
	The motor moves to position 2000, waits 2 seconds and moves to position 0. The wait command starts if the motor is inside the set window.

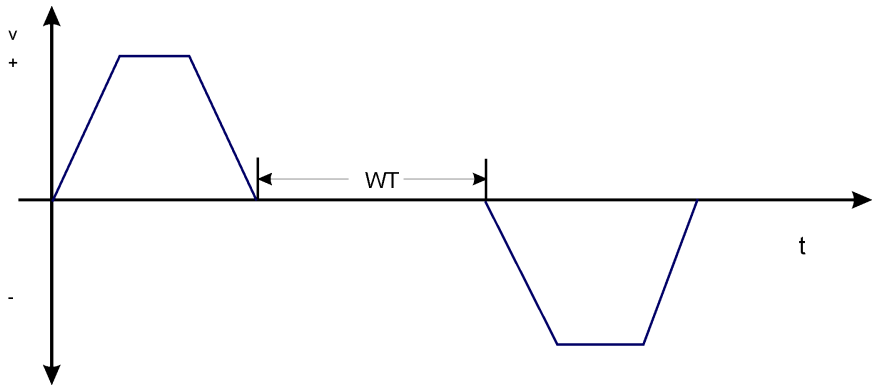


Figure 24: Wait time

WI[t.g:]n± Wait for input line

Range for t: 0 to 59 (depends on the number of nodes)
Range for g: 1 bis 3 (PMCprimo Drive/2)
Range for n: 1 bis 8

This command tells PMCprimo to wait until the specified input line goes to the specified state. If the input line is already in that state then the "**WI**" command terminates immediately. An input line that is defined for an other function may be used in a "**WI**" command. If there is no value given for g, PMCprimo uses the inputs from input byte 1. If there is no value given for the node number t, PMCprimo uses the actual node number.

Example state input 2, byte 2:, node 0:

PMCprimo User	Comment

`0.1>  MA500;WI2:2-;MA0<CR>`
`0.1M`
`0.1W`
`0.1M`
`0.1>`
	The motor moves to position 500, waits for the input state and moves to position 0.

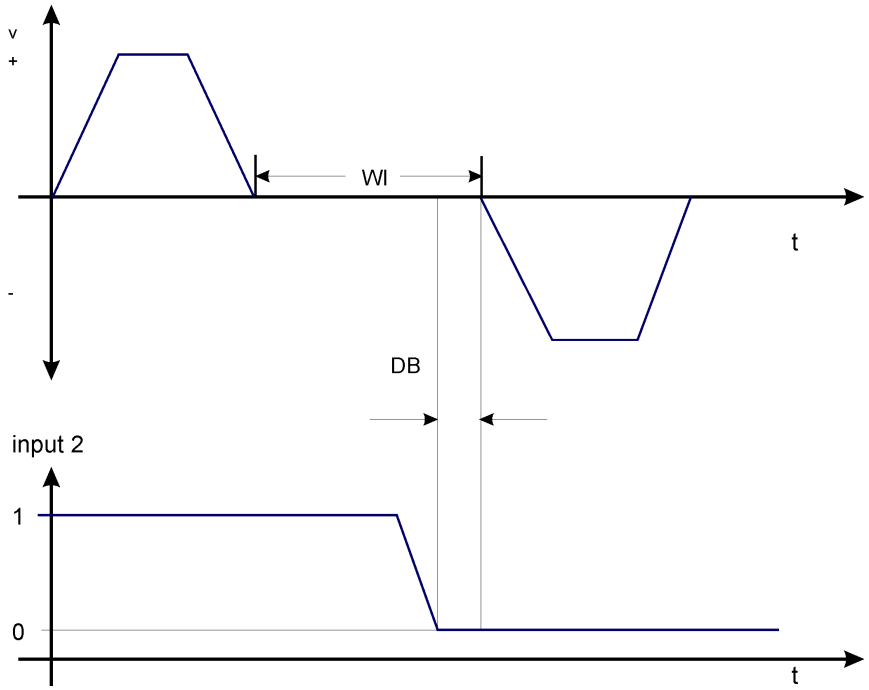


Figure 25: Wait for input line

WA±nn Wait for absolute position

Range: ±4.000.000
As of version **2.004** new range for n: ±2.000.000.000

This command tells PMCprimo to wait until the current channel reaches the given absolute position before executing the next command. The position is specified in increments.

Example: Change of velocity after reaching a specified position:

PMCprimo User	Comment
`0.1>  SV200;VC+;WA1500;SV100<CR>`	
`0.1M`	The velocity is initially 200 counts/sec. The motor moves
`0.1W`	to position 2000 and at position 1500 the velocity is
`0.1M`	changed to 100 counts/second
`0.1>`	

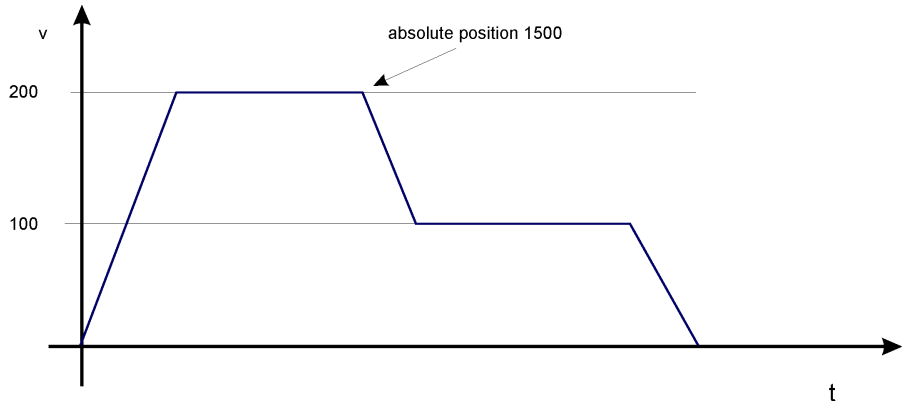


Figure 26: Wait for absolute position

WR±nn Wait for relative position

Range: ±8.000.000
As of version **2.004** new range for n: ±2.000.000.000

This command is similar to the "**WA**" command above. It tells PMCprimo to wait until it reaches the specified position relative to some previous position. The position is specified in increments.

The wait relative command starts a "**WR**" position counter. If the position counter reaches the value of nn, the "**WR**" command is finished.

Example 1: Change of velocity after reaching a specified position:

PMCprimo User	Comment
`0.1>  VC+;WR5000;SV1000;WR2000;ST<CR>`	
`0.1V`	The motor moves with constant velocity 5000 counts,
`0.1W`	changes the velocity, moves 2000 counts and
`0.1V`	stops the motor
`0.1S`	
`0.1>`	

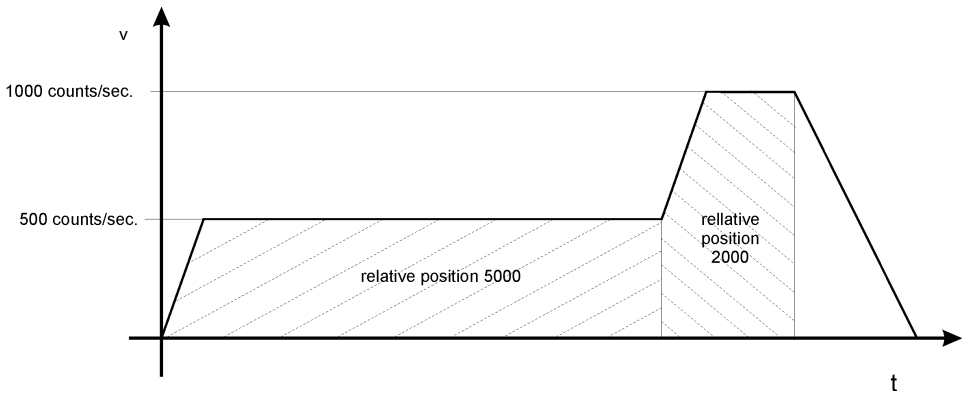


Figure 27: Wait for relative position

WF Wait for reference input

This command sets PMCprimo into the wait state, until a reference input is seen. It may be useful in sequences, to allow the reference action to be changed after detecting the first reference since PMCprimo was started. If no reference input or marker input is defined, then the "**WF**" command returns the error message „no reference input defined"

Example waiting for reference input:

PMCprimo User	Comment
`0.1>  CH1;RW100001;IN+;WF;RW1<CR>`	
`0.1I`	The motor initialises in positive direction and corrects only the position counter and not the motor position. After initialisation every reference signal corrects the motor position.
`0.1W`	
`0.1>`	

WB Wait for bound position

This command tells PMCprimo to wait until the motor passes the next bound position (positive or negative) before continuing with the command string (see also "**SB**" command page 108).

Example:

PMCprimo User	Comment
`0.1>  XM LINEAR;WB;ST<CR>`	The slave is executing a linear position map. After the slave reaches its bound the stop command is executed.
`0.1X`	
`0.1W`	
`0.1S`	
`0.1>`	

WSn Wait for motor status

Enhancement as of **version 1.010**

With this command PMCprimo waits for the next command until the motor status n happens (possible values for motor status see capture 9.3).

Example wait for motor status "**PC**":

PMCprimo User	Comment
`0.1> mr100000`	Set motorstatus to posistioning command
`0.1>`	
`0.1M ws0;sv`	Wait for status "**PC**" and return value for "**SV**"
`0.1M`	
`0.1>`	Status PC was reached
`0.1 SV=5000`	Output of "**SV**"

WC±nn Wait for bound overflow count

Range: ±2.000.000.000

This command tells PMCprimo to wait until the bound overflow counter has changed by the specified count value before continuing with the command string. It may be used, for example, to wait for a given number of machine cycles to complete before stopping.

Example: Wait for bound overflow count:

PMCprimo User	Comment
`0.1>  VC+;WC10;ST<CR>`	
`0.1V`	The motor moves in positive direction until the bound
`0.1W`	overflow counter reaches the value of 10 and then the
`0.1S`	motor stops.
`0.1>`	

WE End wait state

This command ends the current wait state as completed normally. This allows the user to escape from a wait state early but to continue with commands following the wait command. The "**GE**" command ends all wait states (see page 63).

Example end wait state:

PMCprimo User	Comment
`0.1>  ES P_WE`	Sequence makes the command "**WE**"
`0.1>  CH0.1; WE`	
`0.1>  NS`	
`0.1>  DI1:1-,P_WE<CR>`	End wait state if input becomes low.
`0.1>  MA1000;WI1:2+;MA0<CR>`	
`0.1M`	Motor one moves to position 1000,
`0.1W`	waits until I1:2 is high, or input 1 sends end wait state
`0.1M`	motor one moves to position 0.
`0.1>`	

8.8 Error Trapping

PMCprimo continuously monitors various aspects of its performance, in order to detect a range of error conditions. Some errors are critical, in that they prevent PMCprimo from controlling the motor correctly, or they indicate some external failure such as an encoder wiring fault. Others may be more or less important depending on the application.

Critical errors are called „motor off“ errors. If such an error is detected, the axis shuts down to the motor off state, with the motor enable relay switched off. This is the safest course of action for the system to take when these error conditions occur. An error message is output on the main serial port, to indicate which error has been detected. An error code is also shown on the LED display. The motor error conditions are as follows.

- Motor position error
- Motor timeout
- Limit switch input detected

The following error conditions may also be enabled as motor off errors, by setting bits in the error options word “**EW**” (see page 96). When enabled, these also cause the axis to shut down to the motor off state. If not enabled as a motor off error, they are treated as user errors and just give an error message on the serial port. These optional motor off errors are as follows.

- Reference timeout
- Reference outside limits
- Reference correction overrun

SEnn Set maximum position error

Range: 1 to 65535
Default: 1000

This command sets a maximum position error which is continuously monitored by PMCprimo. If the position error at any time exceeds this value, PMCprimo gives a „motor position error“ message and enters the motor off state. PMCprimo must be returned to position control mode before any further motion commands are accepted by PMCprimo. See the Mode commands page 27 for details of the “**MO**” motor and “**PC**” position control commands. The value is defined in encoder counts.

TOnn Set timeout

Range: 1 to 65535
Default: 500

This command sets a monitoring for the encoder signals. The type of monitoring depends on bit 6 of the “**CW**” command (see page 48):

Bit 6 of CW command is 0:

When PMCprimo expects the motor to move and the encoder position does not change for a period that exceeds the timeout, then PMCprimo prints a „motor timeout“ error message and goes to the motor off state. PMCprimo must be returned to position control mode before any further motion commands are accepted. The value is defined in this case in milliseconds.

Bit 6 of CW command is 1:

When PMCprimo expects the motor to move and the encoder position does not change for a period that exceeds the demand position, then PMCprimo prints a „motor timeout“ error message and goes to the motor off state. PMCprimo must be returned to position control mode before any further motion commands are accepted. The value is defined in this case in increments.

LHnn Set position limit high

Range: ± 4.000.000
As of version **2.004** new range for n: ±2.000.000.000

Default: + 4.000.000
As of version **2.007**: Default: + 4.194.304

This command sets up a user-defined limit position. If at any time the absolute position of the motor exceeds the high position limit, PMCprimo gives the „high position limit exceeded“ error message and goes to the motor off state. This is similar to the action taken on detecting a limit switch input. The value is defined in encoder counts. If the SB set bound value is less than the high limit, then the high position limit checking is disabled, as the absolute position value wraps around to zero at the bound position before reaching the high limit position.

LLnn Set position limit low

Range: ± 4.000.000
As of version **2.004** new range for n: ±2.000.000.000

Default: - 4.000.000
As of version **2.007**: Default: - 4.194.304

This command sets up a user-defined limit position. If at any time the absolute position of the motor is less than the low position limit, PMCprimo gives the „low position limit exceeded“ error message and goes to the motor off state. This is similar to the action taken on detecting a limit switch input. The value is defined in encoder counts. If the “**SB**” set bound value is less than the low limit, then the low position limit checking is disabled, as the absolute position value wraps around to zero at the bound position before reaching the low limit position.

RTnn Set reference timeout

Range: 0 to 255
Default: 0

This command sets up a timeout on the reference input. It is used when PMCprimo is set up for continuous monitoring of the reference input, to give a warning error message if the reference input is not detected. A counter is incremented each time the system passes a position half way between the expected reference positions, and cleared each time a valid reference input is detected. If the counter exceeds the “**RT**” value, PMCprimo gives the „reference timeout“ error message. If bit 0 of the “**EW**” command is set to 1 (see page 96), PMCprimo switches off the motor. This facility is also working if the “**IN**” or “**IB**” command (see page 35-36) is executed. The reference timeout function is disabled by setting it to zero.

Example:

PMCprimo User	Comment
`0.1> RT3<CR>`	If the reference signal is 3 times not detected, PMCprimo
`0.1>`	gives an error message.
`0.1> RT0<CR>`	The reference timeout function is disabled.

EWbb Set error options word

Range: 0 to 1111 1111
Default: 0

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

The error word allows various user and motor error options to be turned on or off. The error word bit functions are described below (bit 0 is right and bit 7 is left).

Bit 0: This bit defines if a reference timeout is treated as an user error or as a motor off (see RT command on page 95).

- 0: The reference timeout is treated as an user error, and PMCprimo simply prints an error message.
- 1: The reference timeout error is treated as a motor error, and PMCprimo goes to the motor off state when a reference timeout occurs.

Bit 1: This bit defines if a reference limit error is treated as a motor error or as a motor off error (see LR command page 112).

- 0: The reference limit error is treated as an user error, and PMCprimo simply prints an error message.
- 1: The reference limit error is treated as a motor error, and PMCprimo goes to the motor off state when it occurs.

Bit 2: This bit defines if a reference correction overrun error is treated as a motor error or as a motor off error.

- 0: The reference correction overrun error is treated as an user error, and PMCprimo simply prints an error message.
- 1: The reference correction overrun error is treated as a motor error, and PMCprimo goes to the motor off state when it occurs.

Bit 3: This bit defines if an analogue input out of limits error is treated as a motor error or as a motor off error.

- 0: The analogue input out of limits error is treated as an user error, and PMCprimo simply prints an error message.
- 1: The analogue input out of limits error is treated as a motor error, and PMCprimo goes to the motor off state when it occurs.

Bit 4: This bit controls if the reference timeout errors (see “**RT**” command page 95) as user errors are displayed. The error message is displayed if the error is a motor off error.

- 0: The reference timeout error is displayed.
- 1: The reference timeout error is not displayed.

Bit 5: This bit controls if the reference out of limits (see “**LR**” command page 112) error as user error is displayed. The error message is displayed if the error is a motor off error.

0: The reference out of limits error is displayed.

1: The reference out of limits error is not displayed.

Bit 6: This bit controls if the reference correction overrun error as user error is displayed. The error message is displayed if bit 2 is set not to 1.

0: The reference correction overrun error is displayed.

1: The reference correction overrun error is not displayed if bit 0 is set to 0.

Bit 7: This bit controls if the analogue input out of limits error (see “**AH**” and “**AL**” command page 178) as user error is not displayed if bit 3 is not set to 1..

0: The analogue input out of limits error is displayed.

1: The analogue input out of limits error is not displayed.

8.9 Gain commands

The motor control system operates by sampling the position of the motor at regular intervals, and calculating a motor demand signal according to some control algorithm. The algorithm used is of the following form.

$$V_{Demand} = KP\ e_i + KI\ \Sigma e_i + KD\ (e_i - e_{i-1}) + KV\ (p_i - p_{i-1}) + KF(d_i - d_{i-1}) + KA[(d_i - d_{i-1}) - (d_{i-1} - d_{i-2})]$$

KP = proportional gain constant
KI = integral gain constant
KD = differential gain constant
KV = velocity feedback gain constant
KF = velocity feed-forward gain constant
KA = acceleration feed-forward gain constant
e_i = position error (= demand position - measured position)
d_i = demand position
p_i = measured position

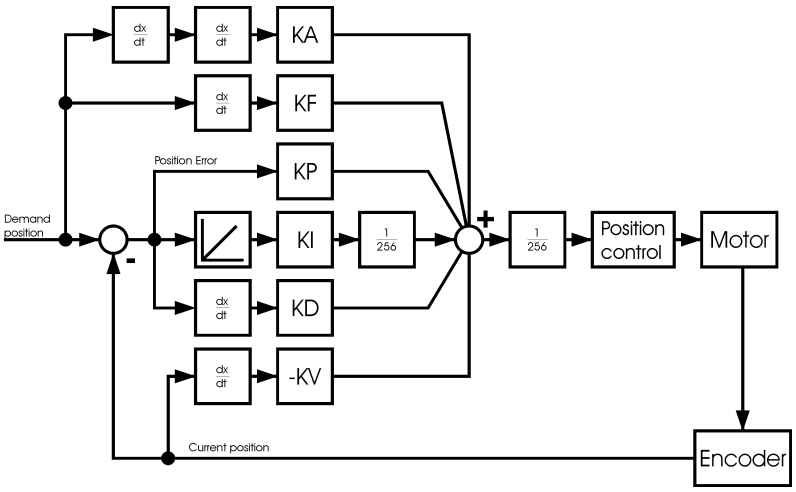
The dynamic behaviour of the motion control system on these gain constants, and on the mechanical characteristics of the system being controlled. Tuning the control system to get best performance on a particular mechanical setup requires setting up these gain constants.

The actual scaling between position error and output voltage, for proportional gain only, is as follows:

$$V_{Soll} = error * KP/256 * 10/2048$$

Where KP is the proportional gain term, and error is the position error, measured in encoder counts. The other control terms are similar.

The performance of any axis may be monitored by using another channel's analogue output as a monitor output. Commands are provided to output various signals on this channel for viewing on an oscilloscope or chart recorder. These are described at the end of this section. This scaling of the monitor output is similar to that of the main demand output, but uses the KM monitor output gain.



KPnn Set proportional gain constant

Range: 0 to 65535
Default: 100

This command sets the proportional gain of the motion control system. The proportional gain acts on the measured position error, which is calculated as the difference between the current demand position and the position measured by the encoder. High gain gives the system a faster response and tighter position control, but if the gain is too high the system may oscillate. For best results, the proportional gain should be set as high as possible without inducing several overshoot or oscillation.

KInn Set integral gain constant

Range: 0 to 65535
Default: 0

This command sets the gain for the integral term in the controller transfer function. When integral control issued, PMCprimo integrates the position error by adding the current error to a running total. Integral gain is useful to remove a constant position error, due to a steady load or friction, or in steady-state velocity control, but also tends to make the motion control system overshoot the target position at the end of a move because of the error accumulated during the move. This problem is known as „wind-up“.

KDnn Set differential gain constant

Range: 0 to 65535
Default: 0

This term uses the differential of the position error (=rate of change of error), which presents the velocity error of the motion control loop. This is useful where the position error is changing rapidly, for example if the required motion is a step change in position.

KVnn Set velocity feedback gain constant

Range: 0 to 65535
Default: 0

This command sets the velocity feedback gain constant. PMCprimo uses the measured position to calculate the motor velocity, scaled by KV, is used in the controller transfer function. Note that differential control uses the rate of change of error, while velocity feedback uses the rate of change of position. Adding velocity feedback is similar to the effect of a tacho generator connected externally to the motor drive, in that it adds damping into PMCprimo. This allows higher values of proportional gain to be used without giving excessive overshoot or oscillation, thus improving the speed of response of PMCprimo.

KFnn Set velocity feed-forward gain constant

Range: 0 to 65535
Default: 0

This command allows the user to set the gain for the velocity feed-forward term in the controller transfer function. It uses the demand velocity as opposed to the measured velocity, and is particularly useful when following a set position or velocity profile. If PMCprimo is using proportional gain only, then there will be a steady position error when running at constant velocity, known as velocity lag. The feed-forward gain has the effect of reducing the velocity lag by adding a component dependent on the demand velocity into the demand signal output. The velocity lag error may be easily reduced to zero or even made negative, by increasing the value of the feed-forward gain. Alternatively, velocity lag may be reduced to zero by the use of integral gain, but this has other effects as well.

For PMCprimo Drive there is a algorithm to get the right "**KF**" value:
KF = 15360 * 1000 / resolution per rotation

Example: resolution 4096 increments/rotation -> KF = 3750

KAnn Set acceleration feed-forward gain constant

Range: 0 to 65535
Default: 0

This command allows the user to set the gain for the acceleration feed-forward term in the controller transfer function. It uses the demand acceleration as opposed to the measured acceleration, and is useful when following a set position or velocity profile. The effect of "**KA**" is to provide a component of the output signal proportional to the required demand acceleration.

SFn Set monitor output function

Range: 0 to 15
Default: 0

This command selects a particular control value to output on the auxiliary analogue output channel. The possible monitor output functions and their associated commands where applicable are as follows.

A change of the control value about 1 is equivalent to a voltage fluctuation of 4,88 mV

n	Function	accordingly command
0	no output function	
1	Demand velocity control value = (Demand velocity * KM * 15) / 1000000+ OM	
2	Measured velocity control value = (Measured velocity * KM * 15) / 1000000+ OM	
3	Position error control value = (FE * KM * 4) / 1000 + OM	
4	Integral of error control value = ((Measured velocity -Demand velocity)*KM*15)/1000000+OM	
5	Absolute demand position control value = (DD * KM * 4) / 1000 + OM	DD
6	Absolute measured position control value = (DP * KM * 4) / 1000 + OM	DP
7	actual measured speed control value = (DV * KM * 15) / 1000000+ OM	DV
8	Master speed control value = (Master speed * KM * 15) / 1000000+ OM	
9	averaged master speed control value = (averaged master speed * KM * 15) / 1000000+OM	
10	Tension control set point control value = AC+ OM	AC
11	Reference error control value = (DF * KM * 4) / 1000 + OM	DF
12	Snapshot position control value = (DS * KM * 4) / 1000 + OM	DS
13	Demand velocity including any reference correction control value = (velocity * KM * 15) / 1000000+ OM	

Enhancement as of **version 1.008:**

14	Velocity ratio of master/slave. Therefore the average velocities are used. The ratio is calculated as follow: control value = DV (Slave) * KM / DV (Master) + OM	

KM±nn Set monitor output gain

Range: ±65 535
Default: 1

This command sets the gain for the monitor output signal. The monitor output functions are scaled by the monitor gain, and not by the gains used in the control algorithm. The signal can be inversed with a changed sign.

OM±nn Set monitor output offset

Range: ±32.767
Default: 0

This command allows the auxiliary monitor output to be offset by a fixed voltage.

Offset in mV = 4,88 mV * OM

Example:

PMCprimo User	Comment
`0.1>  OM0<CR>`	No offset
`0.1>  KM500<CR>`	monitor gain set to 500 (or 1000)
`0.1>`	for measuring with an oscilloscope

Typical monitor outputs from a poorly tuned system:

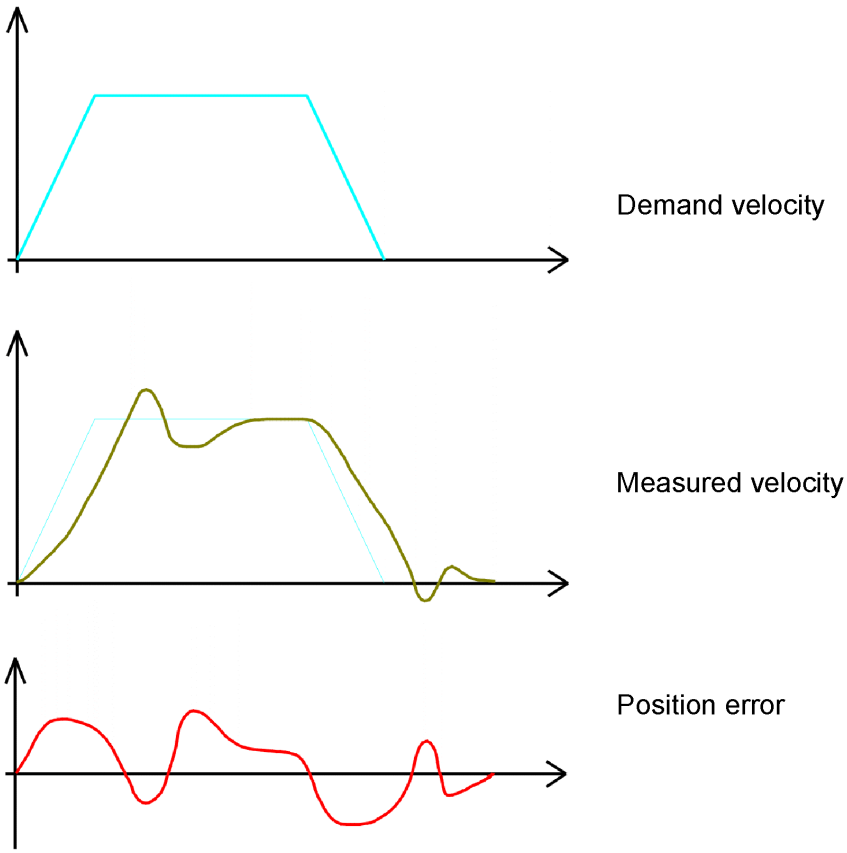


Figure 28: Monitor output functions (this diagram is not to scale)

AO[t.n] Set auxiliary output channel

Range for t: 0 to 59 (depends on number of nodes)
Range for n: 0 to 2
Default: 0

This command allocates the monitor signal for the current channel to one of the analogue outputs on the same board. The monitor function "**SF**" may be defined for the current channel at any time. The "**AO**" command is not available on different nodes. The auxiliary output channel may be set for a particular analogue output only when its channel is in the motor off or virtual motor modes. If the channel is in any other state then the analogue output is not available for use as a monitor output. Conversely, if the analogue output has been allocated to a channel as a monitor signal, then this channel cannot be taken out of motor off or virtual mode.

To return analogue output channel to normal operation, use "**AO0**" on the channel where the auxiliary output is defined. Note that it is also possible to have the auxiliary output signal allocated to the current channel when it is in virtual mode; the signal does not have to be defined on a different channel's output. This may be useful in open-loop control applications.

Enhancement as of **version 1.008:**

In PMCprimo Drive the command "**AO4**" enables a limitation of the torque. PMCtendo firmware version 3.55 is required and the parameter DILIM must be set to 1 (after this change it must done SAVE and COLDSTART). The value 3280 is equal to the IPEAK value.

Enhancement as of **version 2.006:**

Now a variable could be set with the actual setting of "**AO**".

For PMCprimo Drive/2 follow is essential:

AO1 The motor of the drive is moved with constant speed without position control.
If for example an offset is made with the comand OM this is dircet equivalent to the revolution/min.

AO2 The first analog output of tende-DD4 is used. Therefor the parameter „ANOUT1" must set to 6. An output value of 2047 is equal to +10V.

AO3 The second analog output of tende-DD4 is used. Therefor the parameter „ANOUT2" must set to 6. An output value of 2047 is equal to +10V.

AO4 Torque restriction for the connected motor. An output value of 3280 is equal to "IPEAK"

Example:

PMCprimo User	Comment
`0.1: AO 0.2<CR>`	Define auxiliary output
`0.1: $VAR=AO<CR>`	Assignment to variable
`0.1: $VAR<CR>`	value check
`0.1:`	
`0.1: $VAR=2 (0x00000002)`	

Reference Commands

This section describes the commands available to make use of the position reference facilities. In particular, these commands allow the user to set up a repetitive position reference signal, and to use it to automatically adjust the absolute position of the channel. The position of the encoder is immediately stored when the reference input signal is detected. This position is compared with an expected reference position, either the current zero position or the nearest bound position. The difference is defined as the reference error, and the absolute position may be corrected by this amount if required.

PMCprimo supports two types of reference input signal. The encoder marker signal is connected via the Z and /Z inputs to a dedicated fast reference input, called the zero marker input, which responds to signals down to a minimum pulse width of 60ns. This is fast enough to deal with the single marker pulse from a high resolution encoder running at high speed. This input is configured with the DZ command. In addition to this, reference inputs may be programmed on inputs 1 or 2, using the DR command. These additional reference inputs are intended for use with other devices such as proximity switches or micro switches. Note that the fast reference or zero marker input is only used for the encoder marker signal, and cannot be programmed for any other uses, while inputs 1 to 2 may be programmed for any purpose.

PMCprimo 2+2/16+*:* Inputs 1 – 4 are available as reference inputs.

PMCprimo Drive/2*:* Inputs 1 –2 are available as reference inputs.

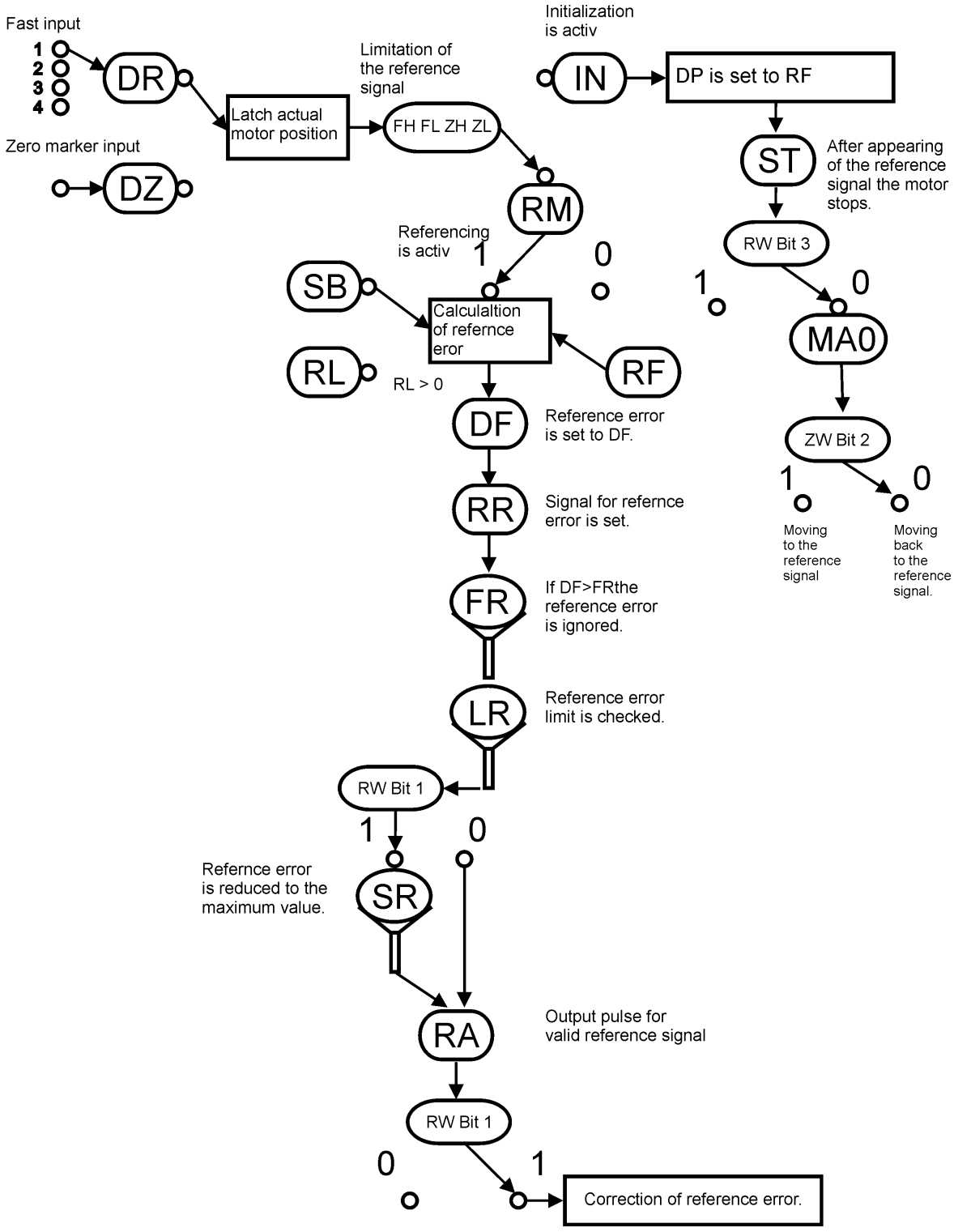


Figure 29: Cooperation of the commands after detecting a reference signal.

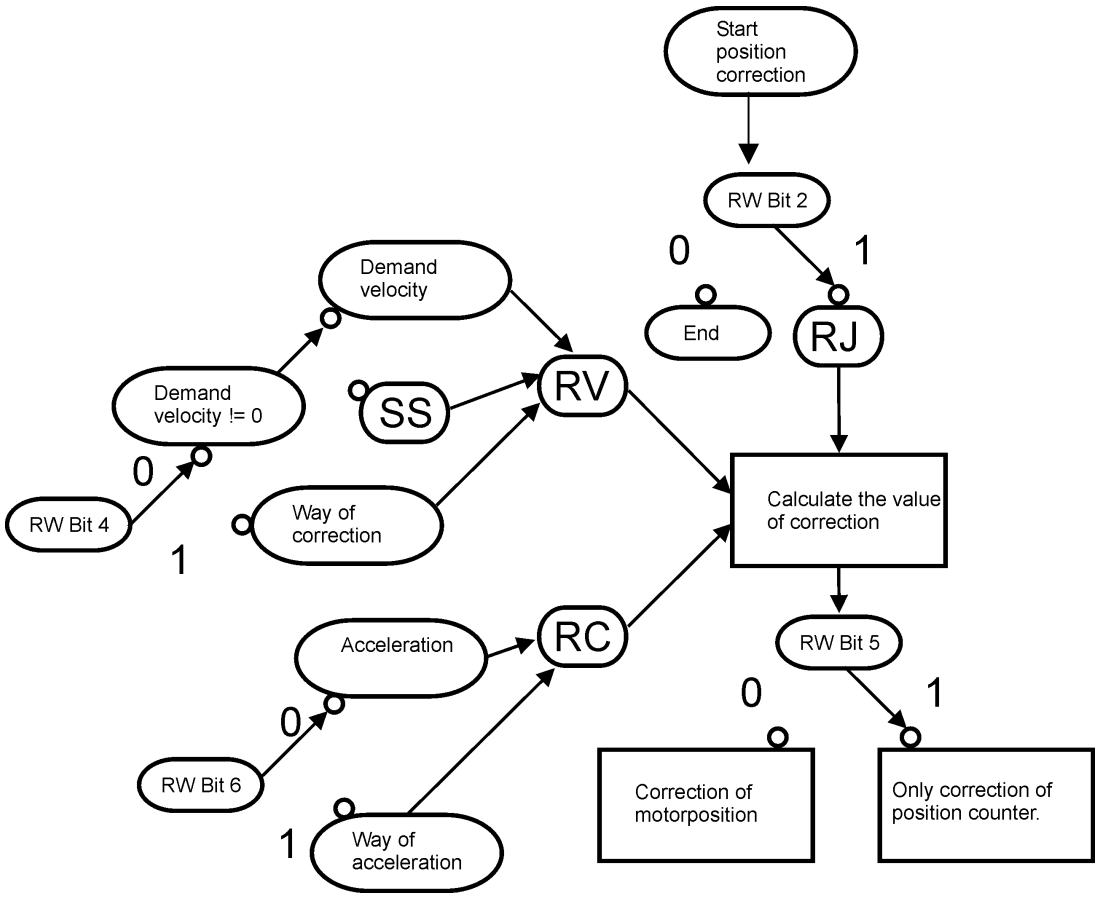


Figure 30: Cooperation of the commands reference error correction

ZC[±nn] Zero position counter or set position

Range nn: ± 4.000.000, or no parameter
As of version **2.004** new range for nn: ±2.000.000.000, or no parameter

If a position value is given, PMCprimo sets the current demand position to the given (absolute) value. If no value is given, the current demand position is set to zero.

Example 1:

PMCprimo User	Comment
`0.1> MA-5000;ZC<CR>`	The motor moves to -5000 and
`0.1>`	than the position counter is set to zero.

Example 2:

PMCprimo User	Comment
`0.1> ZC1000<CR>`	The position counter is set to 1000.
`0.1>`	

SBnn Set position overflow bound

Range: 1 to 4.000.000
As of version **2.004** new range for n: 1 to 2.000.000.000

Default: 4.000.000
As of version **2.005** new default value: 4.194.304
This helps to avoid problems with absolute encoders because therefor the bound must have a value of 2^n.

This command sets upper and lower bounds on the absolute position of the channel. If the position of the motor exceeds the upper bound then the position bound value is subtracted from the current position. If the position goes below the lower bound, the bound value is added to the current position to keep the position within bounds. Note that this does not limit the range of any move commands, but only changes the value of the final position for moves outside the position bounds. This is illustrated by the example below. There is also a 32-bit position overflow counter which is incremented when the position passes the upper bound, and is decremented when the position passes the lower bound. This effectively provides a 32-bit high order extension to the absolute position. The overflow count may be displayed and reset to zero by the "**BC**" command (see page 109).

Example:

PMCprimo User	Comment
`0.1>  SB1000<CR>`	Set position overflow bound to 1000
`0.1>`	

A typical application of this is on a cyclic or rotary machine, where it is required to know the motor position to within one revolution of the motor only, but it is not necessary to distinguish between complete revolutions of the motor. If a move from zero to position 1500 is executed, the final displayed position value is 500. The motor has moved a total distance of 1500 counts as required, but the final position is the remainder when divided by the bound value. If a move from zero to position -1500 is executed, the final position is -500. In this application, the position overflow counter represents the number of complete revolutions of the motor from the zero position to the current position, and the normal position value defines the position within one revolution.

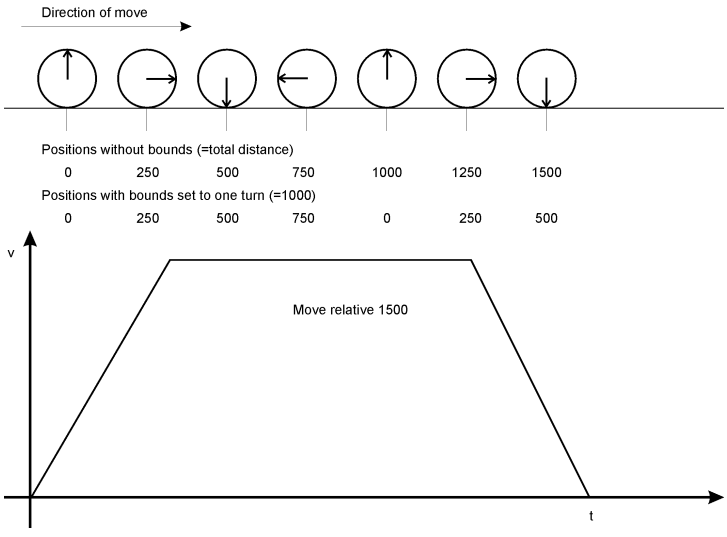

Direction of move
Positions without bounds (=total distance)
0 250 500 750 1000 1250 1500
Positions with bounds set to one turn (=1000)
0 250 500 750 0 250 500
v
Move relative 1500
t

Figure 31: Position bounds

BC±nn Set/display position overflow counter

Range nn: ±2.147.483.647

This command sets the position bound overflow counter to the specified value. The overflow counter is incremented when the position exceeds the upper bound, and is decremented when the position passes the lower bound. If no value is given, the current value of the bound overflow counter is displayed.

RLnn Set reference repeat length

Range: 0 to 4 000 000
As of version **2.004** new range for n: 0 to 2.000.000.000

Default: 0

This command sets the reference repeat length. This is the position at which the channel expects to see the reference position signal. If "**RL**" is set to zero, then the channel uses the bound position, set by the "**SB**", as the expected reference position. If "**RL**" is set to some value greater than zero, then it is used as the expected reference position instead of the bound value. When a reference signal is detected, the position is compared with the nearest multiple of the reference repeat length, instead of to the nearest zero or bound position. This allows the expected reference position to be set independently of the bound position. Note that if "**RW**" bit 7 is set to 1, then the "**RL**" value has no effect and the reference error is calculated relative to the zero position.

A typical example where this is useful is a lead screw application, where the encoder is mounted on the motor and provides a marker signal every turn of the motor, while the bound value must be set larger than the total travel required by the motor. Using the "**RL**" command, the reference repeat length is set to the number of counts per turn of the motor, while the position bound is set as required by the linear motion. Each encoder marker signal detected then gives a useful reference error measurement which may be used for correction if required.

RMn Set continuous reference mode on/off

Range: 0 and 1
Default: 0

This command enables and disables the fast reference input (if defined by the "**DZ**" command), and any reference inputs defined by the "**DR**" command. If "**RM**" is set to 1, all reference inputs are enabled. If it is set to zero, all reference inputs are disabled.

RWbb Set reference options word

Range: 0 to 1111.1111
Default: 0

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command allows various reference functions to be enabled and disabled. The bit functions for the reference word are described below (Bit 0 is right and Bit 7 is on the left side):

Bit 0: This bit enables and disables the position correction on detecting a reference signal, if the reference mode is enabled with the "**RM**" command.

0: Position correction disabled.

1: Position correction enabled.

Bit 1: This bit defines the action taken if the reference error is greater than the maximum value set by the "**SR**" command ("**SR**" command see page 111).

0: A reference error greater than the maximum value is ignored completely and its value is discarded. This also inhibits the „reference out of limits" error message, and does not update the reference error value displayed by the "**DF**" command.

1: If the reference error is greater than the maximum value set by the "**SR**" command, the channel corrects by this maximum value. The „reference out of limits" error is reported if "**LR**" is set to zero, and may set the channel off if required (see "**ME**" command page 63).

Bit 2: This bit enables the adjustment position set by the "**RJ**" command (see page 115).

0: No position for reference correction defined. The correction takes place immediately the reference signal is detected.

1: The channel defers the position correction until the motor passes the adjustment set by the "**RJ**" command.

Bit 3: This bit inhibits the move back to the new zero position in the "**IN**" command (see page 35).

0: The "**IN**" command finishes with a move back to the new zero position defined by the just detected reference input.

1: The move back to the new zero position is inhibited.

Bit 4: Enhancement as of **version 1.008:**
This bit defines the parameter of the command "**RV**" (see page 113)

0: "**RV**" sets the velocity for the reference correction

1: "**RV**" sets the distance of the reference correction. With this function the reference error is spread out to a defined distance. Bit 6 of "**RW**" must be set if bit 4 is set. If not then a warning is given out and bit 6 is set automatic.

Bit 5: This bit defines whether the channel corrects only the displayed position value or the motor and the displayed position.

0: Motor position and displayed position are corrected.

1: Only the displayed position value is corrected.

Bit 6 Enhancement as of **version 1.008:**
This bit defines the parameter of the command "**RC**" (see page 114)

0: "**RC**" sets the acceleration for the reference correction

1: "**RC**" sets an acceleration distance.

Bit 7: not used

SRnn Set maximum reference correction

Range: 0 to 65.535
Default: 0

This command, when set to a non-zero value, limits the maximum allowed reference correction to the specified number of encoder counts. It may be used to eliminate false reference signals at positions far away from the expected reference position, or to allow the position reference facilities to be used even when the machine cycle length is not the same as the distance between reference marker signals.

When a reference signal is seen, the reference error is calculated as the difference between the zero position defined by the reference input, and the zero position or nearest bound position as measured by the normal system encoder counters.

If enabled by bit 0 of "**RW**" and inside the limit defined by the "**SR**" command, the position is corrected by this reference error.

If the reference error is greater than "**SR**", then the action taken depends on bit 1 of "**RW**" and the "**FR**" parameter.

If Bit 1 of "**RW**" is zero and "**FR**" is set to zero, then the position is not corrected, the out of limits reference error value is discarded, and the reference is ignored completely.

If "**RW**" bit 1 is set, or "**FR**" is non-zero, the position is corrected by an amount equal to "**SR**".

When the reference error is larger than "**SR**", the "**LR**" parameter (see page 112) is zero, and correction up to the value of "**SR**" is enabled by "**RW**" bit 1 or "**FR**" non-zero, then a „reference out of limits" error message is given for reference errors larger than "**LR**". If required, PMCprimo may be programmed to generate a motor error when this error is detected. This facility depends on bit 1 of "**EW**" (see page 96).

FRnn Set filter on reference error

Range: 0 to 65535
Default: 0

This command, when set to a non-zero value, defines a maximum value for the reference error in encoder counts. Any reference which gives a reference error value greater than "**FR**" is ignored completely. It is used to eliminate false reference signals at positions far away from the expected reference position. It is independent of the value of "**SR**" (see page 111), which defines the maximum allowed reference correction.

LRnn Set reference error limit

Range: 0 to 65535
Default: 0

This command, when set to a non-zero value, defines a limit value for the reference error in encoder counts. If a reference is detected and gives a reference error value greater than "**LR**", then the „reference error out of limits" error is reported. It is independent of the value of "**SR**" (see page 111), which defines the maximum allowed reference correction, and of "**FR**", the reference error filter value. When "**EW**" bit 1 is set (see page 96), a reference limit error also sets the channel to motor off.

Change as of **version 1.007:**
The monitoring of "**LR**" is made before the "**SR**" monitoring. This means that if "**SR**" > "**LR**" a message occurs. In the versions before "**SR**" was first priority and the reference error was reduced to the maximum value and the "**LR**" monitoring has no effect.

RF±nn Set reference offset

Range: ± 4 000 000
As of version **2.004** new range for n: ±2.000.000.000

Default: 0

This command sets the offset for the reference position. When "**n=0**" PMCprimo regards the position of the reference input signal as absolute position 0. When the value is "**n≠0**" PMCprimo regards the position of the reference input signal as absolute position "**n**".

For FS7 or 8 or FS9 or 10 the RF-value will be recognized. RF-value is subtract from the absolute position. You can correct an offset.

RVn Set reference correction velocity

Range: 0 to 200%
Default: 0

This command sets the correction speed for any reference error. PMCprimo is enabled with a "**RV**" set to a non-zero value to give a limit to the motor velocity during a reference error is corrected. If "**RV**" is set to zero, then the reference error correction is performed immediately in one step. The value for "**RV**" is given as parts per hundred of the actual demand velocity. If a reference error is detected although the motor velocity is zero, the channel will correct the reference error with the velocity given by the "**SS**" command (see page 42). The following algorithm is used if "**n≠0**":

Change velocity = actual velocity x RV/100

If the reference correction velocity is set to small, or the reference error is too large, then it is possible for the next reference signal to arrive before the correction for the previous reference is complete. This condition is called reference correction overrun, and is indicated by the „reference correction overrun" error message. This error may be set to give either a user error or a motor, by setting bit 2 of "**EW**" (see page 96). If this error occurs, it indicates either that the machine is not performing correctly and is giving excessive reference errors, or that the value of "**RV**" is too small and should be increased. The acceleration during reference correction could be set with the "**RC**" command.

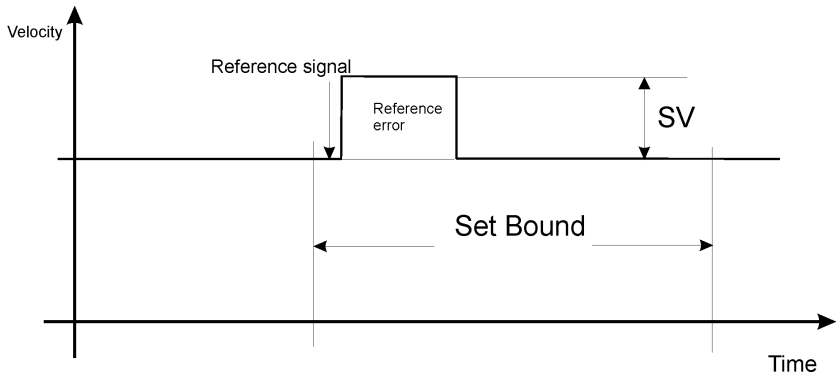


Figure 32: Reference correction with "**RV**"

Enhancement as of **version 1.008:**

If bit 4 of the command "**RW**" (page 110) is set, "**RV**" sets the distance of the reference correction. With this function the reference error is spread out to a defined distance. Bit 6 of "**RW**" must be set if bit 4 is set. If not then a warning is given out and bit 6 is set automatic.

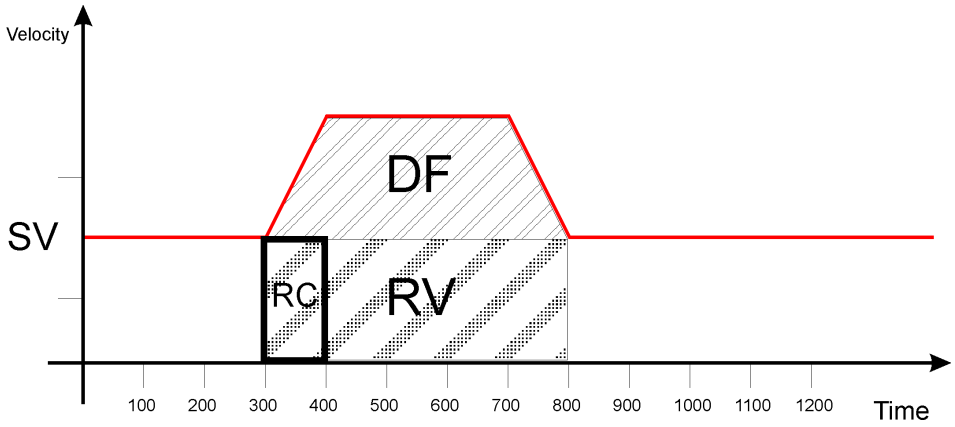


Figure 33: Reference correction with “RV”

The complete distance “**RV**” and the acceleration distance “**RC**” are predetermined. The required acceleration and speed for balancing of the reference error “**DF**” are calculated automatically.

RCn Set reference error adjustment acceleration

Range n: 0 to 2.000.000
Default: 0

The adjustment of a reference correction changes the demand position of a channel. PMCprimo can set an acceleration for the change between the actual velocity and the adjustment velocity. The parameter “**RC**” is given in counts / second^2. If there is no zero parameter given to “**RC**”, the acceleration is not limited.

RJ±nn Set deferred reference adjustment position

Range: 0 to 4 000 000
As of version **2.004** new range for n: ±2.000.000.000

Default: 0

This command allows the position correction on a reference input signal to be deferred until the motor passes a defined position. In some circumstances it may not be desirable to allow a sudden position correction to occur at the reference position, for example because of some mechanical interaction with other parts of a machine. In such a case, the "**RJ**" command defines a position which the motor must pass before the correction due to the reference signal is executed. This function is enabled by bit 2 of "**RW**" (see page 110). If this bit is set to zero, the reference correction takes place immediately.

If the value of "**RJ**" is n times greater than "**SB**" (see page 108) resp. "**RL**" (see page 109) than the correction of referencing will shift for n bounds.

DZb Define zero marker input on/off

Range n: 0 and 1
Default: 0

This command defines whether the encoder zero marker input (the fast reference input) is on or off. The sense of this input is fixed and cannot be programmed. If the value is 1, the zero marker input is turned on.

RN SetReferenceAdvanceFactor

Range n: 0 to 65.535
As of version **2.005** new range for n: ±65.535

Default: 0

This function block is used, to realise a shift dependent on the velocity of the referencing signal. This can be necessary if the used sensor switches only slowly and, therefore, the measured position depends on the velocity.

The calculation occurs according to the following formula (calculation as with "**PA**" and "**BA**"):

$$\text{Displacement} = \frac{\text{demand velocity} * RN}{65536}$$

The value is a time which results from the calculation. 1 ms corresponds to the value 64. As a result the shift can be adjusted in steps of 15,625 micro seconds.

The average of the setpoint velocity is used for the calculation. The time interval set with the function block SetVelocityAveragingTime is used.

FWbb Set reference filter options word

Range bb: 0 to 1111.1111
Default: 0

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command sets the options for the reference filter and for expanded reference options (bit 0 is right and bit 7 left):

Bit 0: This bit enables/disables the filter parameters “**ZH**”, “**ZL**”, “**FH**”, “**FL**”.

0: Parameters “**ZH**”, “**ZL**”, “**FH**”, “**FL**” disabled.

1: Parameters “**ZH**”, “**ZL**”, “**FH**”, “**FL**” enabled.

Bit 1: This bit defines whether the “**RJ**” (see page 115) is executed relative from the reference signal or is executed at the absolute position of the channel.

0: PMCprimo executes “**RJ**” relative to the reference signal.

1: PMCprimo executes “**RJ**” at the absolute position of the channel.

Bit 2: not used.

Bit 3: not used.

Bit 4: not used.

Bit 5: not used.

Bit 6: not used.

Bit 7: not used.

ZHnn **Set reference input true high limit**

ZLnn **Set reference input true low limit**

FHnn **Set reference input false high limit**

FLnn **Set reference input false low limit**

Range: 0 to 65535
As of version **2.004** new range: ±2.000.000.000

Default: 0

These four parameters define limits on the width of the reference input signal seen by the "**DR**" input. They allow the channel to respond to a valid reference signal only if it matches the width limits given. The reference input is accepted as valid only if it is false for a distance which is between "**FL**" and "**FH**", and it is then true for a distance between "**ZL**" and "**ZH**". The input is therefore recognized as valid on the trailing edge of the reference signal, when it switches from true to false. However, the reference error for initialisation and position correction purposes is calculated relative to the leading edge of the input signal, when it switches from false to true, as defined by the "**DR**" command. The distance limits are specified in encoder counts.

To disable any threshold test, set the limit value to zero. For example, to check that the reference input is false for at least 200 counts, with no maximum false distance, set "**FL**" to 200 and "**FH**" to 0. To disable reference width checking completely and return to normal operation, set all four values to zero.

This feature is used to pick out a valid reference pattern from other signals that may be detected by the reference sensor. It is particular useful in printing and registration applications.

Summary:

Reference input is accepted if

- ZL <= distance with "**DR**" input true <= "**ZH**" and
- FL <= distance with "**DR**" input false ("**DR**") <= "**FH**"

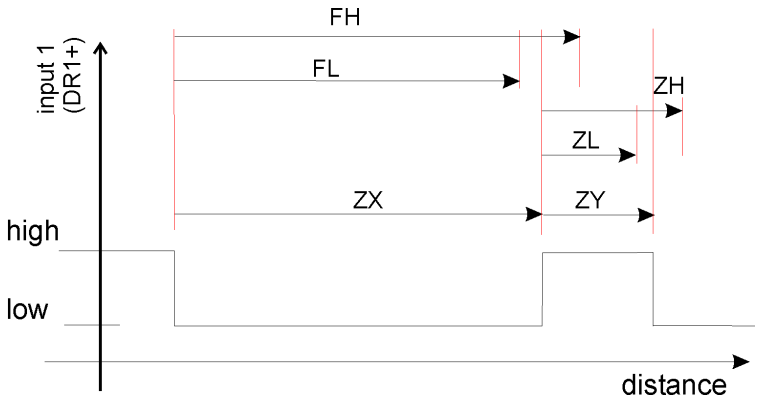


Figure 34: Reference width checking with ZH, ZL, FH and FL

ZX Display length reference signal false

This command shows the measured distance of the reference signal is false. The value will only be displayed if bit 0 of “**FW**” is set to 1. If bit 0 of “**FW**” command (see page 116) is set to 0, the “**ZX**” is also set to 0 automatically.

ZY Display length reference signal true

This command shows the measured distance of the reference signal is true. The value will only be displayed if bit 0 of “**FW**” is set to 1. If bit 0 of “**FW**” command (see page 116) is set to 0, the “**ZY**” is also set to 0 automatically.

RA[g:]n± Define reference accepted output

Range for g: 1 to 2 (PMCprimo Drive)
1 to 3 (PMCprimo 2+2)
1 to 20 adjustable with command "**CD**" (PMCprimo 16+)
1 to 10 adjustable with command "**CD**" (PMCprimo Drive2)
Range for n: 1 to 8

This command defines the specified output line as a reference accepted output. The output gives a 1ms pulse when a reference input signal has been accepted as a valid reference. The ±sign gives the low or high level. An unsigned command deletes an existing definition. This may be used to indicate the presence of a good product, for example when using reference width checking with the commands "**FH**", "**FL**", "**ZH**", "**ZL**" (see page 117).

Enhancement as of **version 1.006d**:
It is also given pulse when initialisation with "**IN**" or "**IB**" (see page 35 and 36) is executed.

RHnn Set reference holdoff time

Range: 0 to 10.000
Default: 0

This command sets up a holdoff time on the reference input. When "**RH**" is set to a non-zero value, the reference holdoff function is enabled. When a reference input is detected, no further reference inputs are recognized until the holdoff time has elapsed. The reference holdoff function is disabled by setting "**RH**" to zero.

OR[g:]n± Define output for reference adjustment

Range for g: 1 to 2 (PMCprimo Drive)
1 to 3 (PMCprimo 2+2)
1 to 20 adjustable with command "**CD**" (PMCprimo 16+)
1 to 10 adjustable with command "**CD**" (PMCprimo Drive2)
Range for n: 1 to 8

This command defines the specified output line to be true when reference adjustment is taking place. An unsigned command deletes an existing definition.

Change as of **version 1.008a**:

The output is set for 1 millisecond to 1, also the reference error is 0. In older versions the output was not set in this case.

JF[g:]n± Reference adjustment forwards output

Range for g: 1 to 2 (PMCprimo Drive)
1 to 3 (PMCprimo 2+2)
1 to 20 adjustable with command “**CD**” (PMCprimo 16+)
1 to 10 adjustable with command “**CD**” (PMCprimo Drive2)
Range for n: 1 to 8

This command defines an output line which indicates when a reference adjustment in the forward direction is taking place.

JB[g:]n± Reference adjustment backwards output

Range for g: 1 to 2 (PMCprimo Drive)
1 to 3 (PMCprimo 2+2)
1 to 20 adjustable with command “**CD**” (PMCprimo 16+)
1 to 10 adjustable with command “**CD**” (PMCprimo Drive2)
Range for n: 1 to 8

This command defines an output line which indicates when a reference adjustment in the backwards direction is taking place.

RKt.n,p Reference position strange channel

Range for node t: 0 to 59 (max. 60 nodes)
Range for channel n: 1 to 3 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 20 adjustable with command “**CD**” (PMCprimo 16+)
1 to 10 adjustable with command “**CD**” (PMCprimo Drive2)
Range for position p: ±4.000.000 (as of version **2.004** new range for p: ±2.000.000.000)

This command makes the reference adjustment of the actual channel at the position 'p' of channel 't.g'. The actual channel waits until channel 't.g' has reached the position 'p' and starts after it with reference adjustment. “**RJ**” (see page 115) at the actual channel is in this case without any effect. The command without 'p' deletes an existing definition.

DR[t.g:]n± Define reference input

Range for t: 0 to 59 (depends on number of nodes)
Range for g: 1
Range for n: 1 to 2 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 4 adjustable with command "**CD**" (PMCprimo 16+)
1 to 4 adjustable with command "**CD**" (PMCprimo Drive2)

This command defines the specified input line as a position reference input for the channel. The sign defines which logic transition is used as the reference position. The channel looks for the specified change in the reference input when the IN initialise position command is executed, and when the automatic reference functions are enabled by the "**RM**" and "**RW**" commands. Note that only input 1 and 2 of byte 1 may be defined as a reference input. If no node number is given, PMCprimo sets t = 0.

Example 1:

PMCprimo User	Comment
`0.1>  DR1:1-<CR>` `0.1>`	This command line defines input 1 of byte 1 as reference input. The falling edge is used as reference signal.

Example 2:

PMCprimo User	Comment
`0.1>  CH2;DR1:2+<CR>` `0.2>`	This command line defines input 1:2 as reference input of channel 2. The rising edge is used as reference signal.

Example 3:

PMCprimo User	Comment

`0.1>  LI1<CR>`
`# Inputs 0.1`
`0.1:1 -R1`
`0.1:2 +R2`
`0.1:3`
`0.1:4`
`0.1:5`
`0.1:6`
`0.1:7`
`0.1:8`
`0.1>`
	This command line lists the input definitions of byte 1

8.10 Digital Inputs and Outputs

PMCprimo has a number of digital inputs and digital outputs on each unit. This section describes the commands available to read the inputs and control the outputs. They may be used immediately at the prompt, or in command strings and sequences.

The input and output lines on PMCprimo are divided into bytes of eight lines, and all commands for the input and output lines take a node and byte number prefix before the line number. If no node number is given, PMCprimo takes the actual node number. Every line address must have a prefix with the byte number.

PMCprimo Drive/2:	2 real input bytes and 1 virtual input byte 1 real output byte and 1 virtual output byte
PMCprimo 2+2:	2 real input bytes and 1 virtual input byte 2 real output bytes and 1 virtual output byte
PMCprimo 16+:	2 real input bytes and 1 virtual input byte 2 real output bytes and 1 virtual output byte

Example:

`SO1:2`	Set output 2 in byte 1
`SO1`	Set all unused outputs in byte 1
`CO2:5`	Clear output 5 in byte 2 (virtual output in PMCprimo Drive)
`RI1`	Read inputs byte 1
`MI3:7`	Mask input 7 in byte 3 (virtual input)
`EI1`	Enable all inputs in byte 1

Virtual In- and Outputs:

PMCprimo Drive:	Input byte 3 and output byte 2
PMCprimo 2+2:	Input byte 3 and output byte 3
PMCprimo 16+:	Input byte 3 and output byte 3

The virtual output lines in byte 2 (PMCprimo Drive) are connected with software to the virtual input lines in byte 3. Except “**DR**” and “**PS**” all commands are available for this I/O-lines.

For the digital output are timer and counter functions available.

Enhancement as of **version 1.008:**

The new virtual input ports 0.101 to 0.1008 are connected to the bus variables $B101 to $B108. A value set to $B101 for example $B101=0xff set all inputs of port 0.101. The inputs have the same functionality as the CAN-inputs. There exist no corresponding outputs like the classic virtual inputs. The values of $B101 to $B108 cant be saved with ‘**SP**’. They can’t be defined as trigger variables. The address range of the modbus was increased for this. The variables cant be used like MA $B101 (no global variables).

SO[t.g:][n][X1,X2] **Set output line n in byte g**

Range for t:	0 to 59, (depends on number of nodes) or no value
Range for g:	1 to 2, or no value (PMCprimo Drive/2) 1 to 3, or no value (PMCprimo 2+2) 1 to 3, or no value (PMCprimo 16+)
Range for n:	1 to 8, or no value
Range for X1:	1 to 8, or no value
Range for X2:	0 to 256, or no value

This command sets the specified output line to a logic high. The output state is maintained until superseded by another command for the same output line. If the specified output is programmed for some defined function, then the „line already defined“ error is reported. If no value is given for “**n**” all unused outputs in byte “**g**” are set to a logic high. If no value is given for “**g**”, PMCprimo uses byte 1.

With the parameters X1 and X2 the output lines n to X1 can set to the value of X2.

Example 1:

PMCprimo User	Comment

`0.1> so0.1:2,4,0x7<CR>`
`0.1> ro1`
`0.1>`
`# Outputs 0.1`
`8 7 6 5 4 3 2 1`
`0 0 0 0 1 1 1 0`
	The output lines 2 to 4 from output byte 1 are set to 1 (0x7).

Example 2:

PMCprimo User	Comment
`0.1> SO1:3<CR>` `0.1>`	This command line sets output 3 in byte1 to a logic high.

Example 3:

PMCprimo User	Comment

`0.1> SO1<CR>`
`0.1>`
`0.1> SO<CR>`
	This command line sets all unused outputs in byte1 to a logic high.
	This command line sets all unused outputs to a logic high.

CO[t.g:][n] Clear output line n in byte g

Range for t: 0 to 59, (depends on number of nodes) or no value
Range for g: 1 to 2, or no value (PMCprimo Drive/2)
1 to 3, or no value (PMCprimo 2+2)
1 to 3, or no value (PMCprimo 16+)
Range for n: 1 to 8, or no value

This command clears the specified output line to a logic low. The output state is maintained until superseded by another command for the same output line. If the specified output is programmed for some defined function, then the „line already defined“ error is reported. If no value is given for “**n**” all unused outputs in byte “**g**” are cleared to a logic low. If no value for “**g**” is given, PMCprimo uses byte 1. The virtual outputs are connected inside to the virtual input lines.

Example 1:

PMCprimo User	Comment
`0.1> CO0.2:7<CR>`	This command clears output line 7 in byte 2 in node 0
`0.1>`	to a logic low.

Example 2:

PMCprimo User	Comment
`0.1> CO1<CR>`	This command sets all unused outputs in byte 1 to a logic low.
`0.1>`	
`0.1> CO<CR>`	This command sets all unused outputs to a logic low.

RI[t.]g Read input lines in byte g

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3, or no value (3 is virtual)

This command reads the current state of the specified input byte. If no parameter is given for g, PMCprimo reads the state of the inputs in byte 1 additional to the logic state of the input lines PMCprimo shows whether the input line is enabled (E), masked (M) or inhibited (B).

Example:

PMCprimo User	Comment
`0.1> RI0.1<CR>`	This command reads the state of the input lines in byte 1.
`1 2 3 4 5 6 7 8`	input line number
`0 1 1 0 0 1 0 0`	Logic state
`E E B M E B E E`	Enabled (E), masked (M), inhibited (B) inputs
`0.1>`	

RO[t.g] Read output line states in byte g

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, (PMCprimo Drive/2) or no value
1 to 3, (PMCprimo 2+2) or no value
1 to 3, (PMCprimo 16+) or no value

This command reads the current state of the specified output byte. If no parameter is given for "**g**", PMCprimo reads the state of the outputs in byte 1. If no parameter is given for "**t**", PMCprimo reads the outputs of node 0.

Example:

PMCprimo User	Comment
`0.1>  RO0.1<CR>`	This command reads the state of the output lines in byte 1.
`1 2 3 4 5 6 7 8`	output line number
`0 1 1 0 0 1 0 0`	Logic state
`0.1>`	

II[t.]g:n± If input true do command line

Range for t: 0 to 59, (depends on number of nodes) or no value
Range for g: 1 to 3 (3 is virtual)
Range for n: 1 to 8

This command allows the user to specify that a command or command line is conditional on the current state of an input line. If the input line specified in the "**II**" command is in the specified state (the condition is true) the remainder of the command line is executed. If the input line is not in the specified state, the remainder of the command line is skipped, and execution proceeds to the next line of input, either the next line of a sequence, or new input commands.

The "**II**" command may be followed by the "**EL**" command. If the input line is true PMCprimo executes the commands following II up to the "**EL**" command or end of line, which comes first. If the input line is false PMCprimo executes the commands following "**EL**" up to the end of line. The "**EL**" command must appear on the same line as the "**II**" command, except in a sequence, where it may also be the first command on the line following "**II**".

Example 1:

PMCprimo User	Comment
`0.1>  II2:2-;CO1:3<CR>`	If input lin2 in byte 2 is low, the output 1:3 is set to logic low.
`0.1>`	

Example 2:

PMCprimo User	Comment
`0.1>  II1:2-;XS PRG3;EL;XS PRG21<CR>`	If input line 2 in byte 1 is low PMCprimo executes PRG3, if input line 2 in byte 1 is high PMCprimo executes PRG21.

IT If motor status true do command line

(**I**f mo**t**or status true do command line)

This command allows the user to specify that a command or command line is conditional on the current motor status. If the motor is switched on (the condition is true) the remainder of the command line is executed. If the motor is off, the remainder of the command line is skipped, and execution proceeds to the next line of input, either the next line of a sequence, or new input commands.

The "**IT**" command may be followed by the "**EL**" command. If the motor is on PMCprimo executes the commands following "**IT**" up to the "**EL**" command or end of line, which comes first. If the motor is off PMCprimo executes the commands following "**EL**" up to the end of line. The "**EL**" command must appear on the same line as the "**IT**" command, except in a sequence, where it may also be the first command on the line following "**IT**".

Example:

PMCprimo User	Comment
`0.1> IT;CO1:3<CR>` `0.1>`	Output 1:3 is set, because the motor is switched on.

IO[t.] g:n± If output true do command line

Range t: 0 to 59, (depends on number of nodes) or no value
Range n: 1 to 8
Range g: 1 to 2 (PMCprimo Drive/2) (2 is virtual)
1 to 3 (PMCprimo 2+2) (3 is virtual)
1 to 3 (PMCprimo 16+) (3 is virtual)

This command allows the programmer to specify that a command or a command line is conditional on the current state of an output line. If the output line specified in the "**IO**" command is in the specified state (the condition is true) then the remainder of the command line is executed. If the output line is not in the specified state, the remainder of the command line is skipped, and the execution proceeds to the next line of input, either the next line of a sequence, or new input commands.

Example:

PMCprimo User	Comment
`0.1> IO2:4+;XS PROG23<CR>`	If the output line 2 in byte 2 is logic high PMCprimo executes PROG23.

MI[t.][g:n] Mask function input

Range t: 0 to 59, (depends on number of nodes) or no value
Range n: 1 to 8
Range g: 1 to 2

This command is used to mask the action of defined function inputs. It allows several input lines to selectively mask out defined actions, depending on the current function activated. For example, a machine start sequence assigned to a function input may mask itself once the machine has started, until the stop sequence assigned to another input re-enables it. This prevents any subsequent signal on the start input from generating unnecessary start sequence commands, which may not be allowed when the machine is running.

Masked inputs are enabled, again by the "**EI**" command. If a "**DI**" input line changes state while it is masked, then the function assigned to the change of state executes when the line is enabled. If the line changes state twice and returns to the same state as when it was masked, then nothing executes when it is enabled.

If a line number is given to "**n**", then the specified line is masked. If a byte number followed by a colon is given with no line number then all function inputs and/or expanded inputs in the specified group are masked. If no line number or byte number is given, then all function inputs in all bytes of the node are masked. If no line number, no byte number and no node number is given, then all function inputs in all bytes in all nodes are masked.

Example:

PMCprimo User	Comment
`0.1>  DI2:1+;PROG15<CR>`	The rising edge of input 2:1 executes PROG15. PROG15 sets Output 1:5, please see the picture below.
`0.1>`	

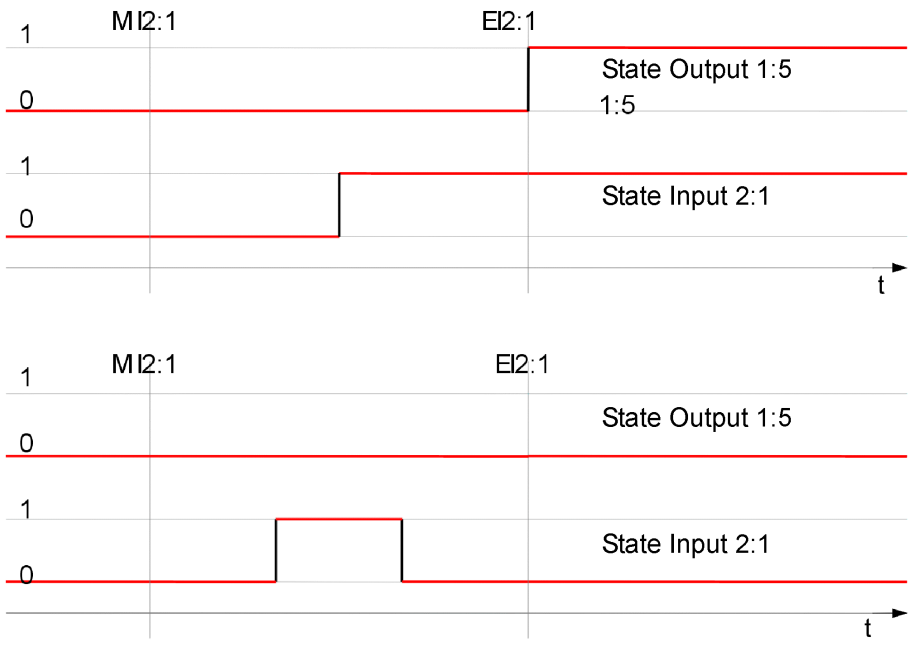


Figure 35: Mask input with MI command

BI[t.][g:n] Inhibit function input

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3, (3 is virtual)
Range n: 1 to 8

This command is used to disable the action of defined function inputs or any expanded input lines. It is similar to the "**MI**" mask input function, but with the difference that all input state changes are ignored on inhibited inputs. Inhibited inputs are enabled again by the "**EI**" command.

If a line number is given to n, then the specified line is inhibited. If a byte number followed by a colon is given with no line number then all function inputs in the specified byte are inhibited. If no line number or byte number is given, then all function inputs in all bytes of the node are inhibited. If no line number, no byte number and no node number is given, then all function inputs in all bytes in all nodes are inhibited.

Example:

PMCprimo User	Comment
`0.1> DI2:1+;PROG15<CR>`	The rising edge of input 2:1 executes PROG15. PROG15 sets Output 1:5, please see the picture below.
`0.1>`	

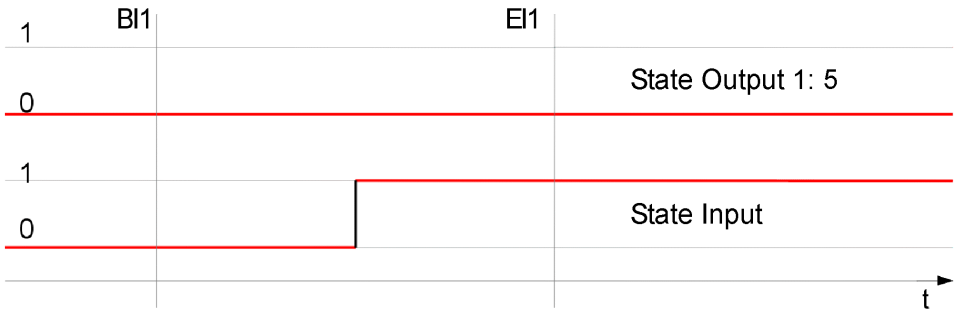


Figure 36: Inhibit input function

EI[t.][g:n] Enable function input

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3, (3 is virtual)
Range n: 1 to 8

This command is used to enable the action of defined function inputs, where they have been masked by the "**MI**" (see page 127) command or inhibited by the "**BI**" command (see page 128). If a byte number followed by a colon is given with no line number then all function inputs in the specified byte are enabled. If no line number or byte number is given, then all function inputs in all bytes of the node are enabled. If no line number, no byte number and no node number is given, then all function inputs in all bytes in all nodes are enabled.

EN[t.][g:n],bb Enable function input with priority

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3, (3 is virtual)
Range n: 1 to 8
Range bb: 0000 0000 to 1111 1111

This command sets the inputs defined in the group '**b**' free like the command '**EI**'. The difference is that only one sequence is executed. If there are more than one sequences to execute of the masked inputs only the input with the lowest input number is executed.

UIt.g:n Undefine input function

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3, (3 ist virtual)
Range n: 1 to 8

This command undefines an existing input definition. It is independent what type of input function is defined. The parameter "**n**" gives the specified input line. If no value for t is given, PMCprimo sets "**t**" = 0. The "**UI**" command expects always a byte and a line number. It is not possible to undefine more when one input function at the same time.

UOt.g:n Undefine output function

Range t: 0 to9, (depends on number of nodes) or no value
Range g: 1 to 2 PMCprimo Drive/2 (2 is virtual)
1 to 3 at PMCprimo 2+2 (3 is virtual)
1 to 3 at PMCprimo 16+ (3 is virtual)
Range n: 1 to 8

This command undefines an existing output definition. It is independent what type of output function is defined. The parameter "**n**" gives the specified output line. If no value for t is given, PMCprimo sets t = 0. The "**UO**" command expects always a byte and a line number. It is not possible to undefine more when one output function at the same time.

TC[t.]g:n[±,X1,X2] **Define timer/counter output**

Range t:	0 to 59, (depends on number of nodes) or no value
Range g:	1 to 2, PMCprimo Drive/2, (2 is virtual)
	1 to 3, PMCprimo 2+2 (3 is virtual)
	1 to 3, PMCprimo 16+ (3 is virtual)
Range n:	1 to 8
Range X1 (count value):	0 to 65535 for modus 0 to 7 and 0 to 31 for modus 8
Range X2 (Mode):	0 to 8

There are several commands available to support timer/counter functions on the digital output lines. The "**TC**" function may be combined with other output functions, for example to provide a defined pulse width for signalling to a PLC, or it may be used with the "**SO**" and "**CO**" commands. If the "**TC**" command is given with the following sign, to define a new timer/counter output, then it should be followed by two parameters. The first parameter is the count value for the timer/counter, and the second value is the timer/counter mode.

The timer/counter mode values are as follows:

Mode	Operation
0	One-shot up counter
1	Cyclic up counter
2	One-shot down counter
3	Cyclic down counter
4	One-shot timer
5	Cyclic timer
6	One-Shot-Timer with restart **(as of version 2.004**)
7	Reserved
8	Shift register

The output line is set true (as defined by the sense in the "**TC**" command) when the timer/counter is first triggered, and it is reset false when the timer/counter reaches its final value, set by the count parameter.

In counter modes 0 to 3, the counter is incremented or decremented when either the "**SO**" command is given for the output, or any other defined output function on the same output goes true. If it is an up counter, its initial value is zero, and its final value is given by the count parameter. If it is a down counter, its initial value is the count parameter, and its final value is zero. On the first count, the output line is set to true, and the counter is started and set to its initial value. When the count is incremented or decremented to its final value, the output line is reset false, and the counter is stopped and reset to its initial value. The counter may be stopped and reset at any time by using the "**CO**" command.

In timer modes 4 and 5, the timer is triggered in the same way as in counter mode, but once triggered it counts once per tick until the final count is reached. The output line is set true when the timer is triggered, and is reset false when the timer reaches the final count.

In one-shot modes 0, 2 and 4, the timer/counter behaves as described above. In cyclic modes 1, 3 and 5, it operates in a slightly different way. When the timer counter reaches its final count, the output line is toggled to its opposite state, the counter/timer is reset to its initial value and continues to run. The output line changes state each time the counter/timer reaches its final count.

In shift register mode 8, the timer/counter behaves as a shift register, to a maximum of 31 bits. The state of the "**TG**" gate input assigned to the timer/counter is clocked into the shift register by an active transition on the TK clock input, or by an "**SO**" command on the "**TC**" output. At the same time, the last bit in the shift register is clocked out onto the "**TC**" output line. An active transition on a "**TZ**" input or a "**CO**" command on the "**TC**" output resets the shift register contents to zero.

Example 1:

PMCprimo User	Comment
`0.2>  TC1:3+,20,1<CR>`	This command line defines a cyclic up counter, which toggles output 1:3 every twenty clocks. SO1:3 increments the counter by 1. CO1:3 stops the counter and resets it to zero.

Example 2:

PMCprimo User	Comment
`0.2>  RA1:4+` `0.2>  TC1:4+,20,5<CR>`	defines a reference accepted output, and adds a one-shot pulse timer to it. The timer triggered by the RA output going true, and then holds the output true for 20ms. Thus it stretches the normal RA output time of 1ms to something a bit longer, to make it easier for it to be seen by a PLC.

Example 3:

PMCprimo User	Comment
`2>  LO1<CR>`	List output definitions in byte 1

```
# outputs 0.1
0.1:1
0.1:2
0.1:3 + TC Count=00020 Mode=1
0.1:4 + RA1 -> TC Count=00005 Mode=4
0.1:5
0.1:6
0.1:7
0.1:8
```

Example 4:

PMCprimo User	Comment
`0.2> TC1:5+,10,8<CR>`	This defines a 10 bit shift register on output 1:5, with the data signal on input 2:1 and the clock signal on input 2:2.
`0.2> TG2:1+,1,5<CR>`	A rising edge on input 2:2 or SO1:5, clocks the current state of input 2:1 into the shift register, and the shift
`0.2> TC2:2+,1,5<CR>`	register updates output 1:5 as appropriate. CO1:5 clears the shift register contents.

Enhancement as of **version 2.004**:

The command "**TC**" was enhanced with a further modus:

Modus 6: One-Shot Timer with restart.

If the defined output is set again with SO n:m and the timer is still running, the timer starts once again. The output is still set TRUE.

LC[t.][g:n] List counter value

(**L**ist **c**ounter value)

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8

The "**LC**" command lists the current count or time value of a timer/counter. This may be particularly useful in applications using a counter to count some external event or signal. If the "**TC**" output is programmed as a shift register, then the current contents of the shift register are displayed, but as a numerical value, not a binary value. It is possible to assign the result of the "**LC**" command into a variable.

TK[t.]g:n[±,g2:n2] **Define timer/counter clock input**

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range n: 1 to 8
Range g2: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n2: 1 to 8

The "**TK**" command defines a clock/trigger input line for a timer/counter output. The parameter g and n gives the input line and g2 and n2 gives the output line.

When a clock input changes from false to true, it increments or decrements a counter, or it triggers a timer, in the same way as the "**SO**" command on the timer/counter output (see "**TC**" command on page 130). When a reset input is true, the timer/counter is stopped and is reset to its initial value, in the same way as the "**CO**" command on the timer/counter output. If a gate input is defined, it allows its timer/counter to run normally when it is true, and holds it at its current value when it is false.

Example:

PMCprimo User	Comment
`2>  TK1:2+,1:1<CR>`	This defines a clock signal on input 1:2, assigned to a timer/counter on output 1:3.
`2>  LI1<CR>`	LI lists this as follows:
`1:1`	
`1:2 + TK -> TC1:1`	
`1:3`	
`1:4`	
`1:5`	
`1:6`	
`1:7`	
`1:8`	

TG[t.]g:n[±,g2:n2] Define timer/counter gate input

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range n: 1 to 8
Range g2: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n2: 1 to 8

The "**TG**" command defines a gate input for a timer/counter. The parameter g and n gives the input line and g2 and n2 gives the output line.

If a position output (see page 144) is defined, the output is only set if "**TG**" is true.

Enhancement as of **version 1.008:**

The command "**TG**" can define a gate input for several outputs. Therefore "**TG**" is assigned to the outputs and is shown with the "**LO**" command (see page 150). To undefine "**TG**", the output must indicate (TG g2:n2).

TZ[t.]g:n[±,g2:n2] Define timer/counter reset input

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range n: 1 to 8
Range g2: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n2: 1 to 8

The "**TZ**" command defines a reset input for a timer/counter. The parameter g and n gives the input line and g2 and n2 gives the output line.

When a reset input is true, the timer/counter is stopped and is reset to its initial value, in the same way as the "**CO**" command on the timer/counter output (see "**TK**" command on page 133).

8.11 Configuration Commands

Configuration commands allow the definition of limit switches and other input definitions. Most of these commands are channel specific!

The number of input and output lines depends on the unit, for example PMCprimo Drive has 12 digital inputs and 8 digital outputs. The input and output lines on PMCprimo are divided into bytes of eight lines, and all commands for the input and output lines take a node and byte number prefix before the line number. If no node number is given, PMCprimo takes the actual number. Every line address must have a prefix with the byte number.

Example:

PMCprimo User	Comment
`0.1: SO1:2`	Output 2 of byte 1 on node 0 set to logic high
`0.1: SO1.1`	Set all unused outputs of byte 1 on node 0 to logic high
`3.1: CO2:5`	Clear output 5 of byte 2 (virtual output on PMCprimo Drive)
`0.1: RI2.1:1`	Read state of inputs from byte 1 on node 2
`0.1: RI0.2`	Read state of inputs from byte 1 on node 0
`0.1: MI3:7`	Mask input line 7 from byte 3 on node 0.
`0.1: EI2`	enable all inputs from byte 2 on node 0

DL[t.] g:n±[,±] Define limit switch input

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range n: 1 to 8

This command defines the specified input line as a limit switch input. The sign defines which logic state represents the out-of-limit condition. When the line goes to the specified state, PMCprimo stops the motor immediately, prints a „limit switch detected“ error message, and goes to the motor off state. A line which has been defined as a limit switch input may be returned to normal operation by entering this command without the sign.

Example:

PMCprimo User	Comment
`0.1>  CH3;DL2:4-<CR>` `0.1>`	This command line defines input 2:4 as a limit switch to channel 3 on node 0. PMCprimo detects a limit switch when the line goes to a logic low.

As of **Version 2.006**:
The command “**DL**” now can set a direction. With this the channel could be moved from the limit switch. The check is done only for one direction.
Example:
DL 0.1:1+,+
With ,+ or ,- the limit switch detection could be done only for the positive or negative direction.
If ,+ or ,- is not set then the function is also changed. In earlier version the motor could not switch on while the limit switch was active. Now the system saves the direction at the time when the limit switch is activated. A movement in the other direction is possible. If the speed is 0 and the limit switch is activated then the function is like in the old versions and “**PC**” is not possible.

Error message:

```
0.1> vc
0.1>
0.1V
0.1:
0.1 High position limit exceeded (20-01-2005 11:57:01)
0.1:
```

PS[t.] g:n± Define position snapshot input

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1
Range n: 1 to 2 (PMCprimo Drive/2)
1 to 4 (PMCprimo 2+2)
1 to 4 (PMCprimo 16+)

This command defines the specified input line as a position snapshot for the channel. The sign defines which logic transition is used to detect the snapshot position. PMCprimo monitors the snapshot input and stores the absolute position value at that time. The snapshot position data may be read at any time by using the "**DS**" command (see page 162). The snapshot function uses the same mechanism as the reference input function to get an accurate measurement of position on an input signal. Note that because of this, only inputs 1 and 2 (3 and 4 only at PMCprimo 2+2 and 16+) in byte 1 may be defined as snapshot inputs. A position snapshot input line may be returned to normal operation by entering this command without the sign.

If "**DR**" or "**DZ**" is already defined for the axis it is not possible to define a position snapshot for the same axis.

Example 1:

PMCprimo User	Comment
`0.2: CH4;PS1:2+<CR>`	This command line defines input 1:2 as a position snapshot for channel 4. The specified input must be on the same node as the specified channel.
`0.4:`	
`0.4: CH4;$VAR=DS<CR>`	The measured value of the position snapshot could read with the "**DS**" command and could be put into a variable.

Example 2:

PMCprimo User	Comment
`0.1: LI1<CR>`	List input definitions byte 1
`# inputs 0.1`	
`0.1:1 undef`	
`0.1:2 + PS4`	Input 2 is position snapshot of channel 4
`0.1:3 undef`	
`0.1:4 undef`	
`0.1:5 undef`	
`0.1:6 undef`	
`0.1:7 undef`	
`0.1:8 undef`	
`0.1:`	
.	

DI[t.] g:n±,Name **Define function input**

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range n: 1 tp 8
Name: Name of sequence

This command defines a specified input line to have the given function. The sign specifies the active state of the input, such that PMCprimo executes the function when the input changes to the specified state. The command function calls a sequence. The sign defines which logic transition is used to execute the sequence. It is possible to define a sequence for the rising edge and a sequence for the falling edge at the same input line. Note that this is not possible if the input line is defined as a reference input or position snapshot. An input function may be returned to normal operation by entering this command without the sign.

Example 1:

PMCprimo User	Comment
`0.2> DI2:2-,TestPrg<CR>`	This command line defines, that the falling edge of input 2:2 starts sequence TestPrg.

Note that it is not allowed to start a running sequence. (Error message ‘Cannot execute sequence <....> while it is in use’)!

MG[t.] g,bb Define input mask group

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range bb: 0000 0000 to 1111 1111 (default: 0)

This command specifies a set of input lines, such that if one line in the set goes active, all the input lines in the set are immediately masked to prevent them acting. The lines remain masked until they are explicitly enabled. The set of lines is specified by a binary parameter where a bit set to 1 includes the corresponding input line 1 in the specified group.

The input mask group should only include inputs defined as function inputs with the "**DI**" command. The "**MG**" command has no effect on other types of input line.

The "**MG**" command applies to the specified input group only. It is not possible to specify automatic masking of inputs in a different input group.

If the state of a masked input changes, the defined input sequence will be start after enable the inputs (see also "**MI**" command on page 127).

Example:

PMCprimo User	Comment
`0.1>  ES PROG1`	
`002E  CH1;VC+`	Move in positive direction
`003E  NS`	
`0.1>  ES PROG2`	
`002E  CH1;VC-`	Move in negative direction
`003E  NS`	
`0.1>  CH1;DI2:1+;PROG1<CR>`	Execute PROG1
`0.1>  CH1;DI2:2+;PROG2<CR>`	Execute PROG2
`0.1>  MG2,00000011<CR>`	Mask group to input 1 and 2 of byte 2
`0.1>`	

In the example above, the input line mask group is defined to include input lines 1 and 2. When input line 1 goes high, inputs 1 and 2 are both masked. If the axis is stopped and the input lines are enabled, it is possible to start the axis next time. This makes sure that only one of the two start functions are enabled at one time.

BG[t.] g:bb **Define input inhibit group**

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range bb: 0000 0000 bis 1111 1111 (default: 0)

This command specifies a set of input lines, such that if one line in the set goes active, all the input lines in the set are immediately inhibited to prevent them acting. The lines remain inhibited until they are explicitly enabled. The set of lines is specified by a binary parameter where a bit set to 1 includes the corresponding input line 1 in the specified group. It is similar to the MG mask group function, but with the difference that all input state changes are ignored on inhibited inputs (see MI command on page 127).

If the state of an inhibited input changes, opposite to MG, the defined input sequence will not start after enable the inputs (see also BI-command on page 128).

Example:

PMCprimo User	Comment
`0.1>  ES PROG1`	
`002E  CH1;VC+`	Move in positive direction
`003E  NS`	
`0.1>  ES PROG2`	
`002E  CH1;VC-`	Move in negative direction
`003E  NS`	
`0.1>  CH1;DI2:1+;PROG1<CR>`	Execute sequence PROG1
`0.1>  CH1;DI2:2+;PROG2<CR>`	Execute sequence PROG2
`0.1>  BG2:11<CR>`	Inhibit group on input 1 and 2 in byte 2
`0.1>`	

The rising flank of input 1 inhibits input 2. The rising flank of input 2 inhibits input 1. Without input inhibit PMCprimo returns an error. With input inhibit no reaction of PMCprimo occurs.

MN[t.] g,bb **Define masked input line**

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range bb: 0000 0000 to 1111 1111 (default: 0)

The command "**MN**" defines an input mask group like "**MG**". The difference is that only the changed input is masked. After the change the input must set free with the '**EI**' command. The right bit is input 1. Every bit is corresponding to an input. Without the parameter n the actual mask is shown.

If the state of a masked input changes, the defined input sequence will start after enable the inputs (see also "**MI**" command on page 127).

Example:

PMCprimo User	Comment
`0.1: ES Prg1`	Sequence Prg1 is defined, which
`002E CH0.1;SV`	prints the value of "**SV**" for test.
`003E NS`	
`0.1:`	
`0.1: ES Prg2`	Sequence Prg2 is defined, which
`002E CH0.1;SA`	prints the value of "**SA**" for test.
`003E NS`	
`0.1:`	
`0.1: DI0.3:1+,Prg1`	Define virtual input 3:1 with sequence Prg1
`0.1: DI0.3:2+,Prg2`	Define virtual input 3:2 with sequence Prg2
`0.1: EI`	Enable all inputs
`0.1: RI0.3`	Show input port 0.3
`0.1:`	
`# Input: 0.3`	
`8 7 6 5 4 3 2 1`	
`0 0 0 0 0 0 0 0`	All inputs are clear.
`E E E E E E E E`	All inputs are enabled.
`0.1:`	
`0.1: MN0.3,11111111`	Every input should masked after its change.
`0.1: SO3:1;WT10;CO3:1`	Set and reset of the virtual output makes a
`0.1:`	signal for the virtual input.
`0.1 SV=20000`	Sequence Prg1 was executed.
`0.1: RI0.3`	Show input port 0.3:
`0.1:`	
`# Input 0.3`	
`8 7 6 5 4 3 2 1`	
`0 0 0 0 0 0 0 0`	
`E E E E E E E M`	The command "**MN**" has effected that input 3:1 was masked.
`0.1: SO3:1`	Another input signal has no effect.
`0.1: EI`	Enable input starts the sequence.
`0.1:`	
`0.1 SV=20000`	Sequence Prg1 was started by "**EI**".
`0.1:`	

BN[t.] g,bb **Define inhibit input line**

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)
Range bb: 0000 0000 to 1111 1111 (default: 0)

The command "**BN**" defines an input inhibit group like "**BG**". The difference is that only the changed input is inhibited. After the change the input must set free with the '**EI**' command. The right bit is input 1. Every bit is corresponding to an input. Without the parameter n the actual mask is shown.

If the state of a masked input changes, the defined input sequence will start after enable the inputs (see also "**MI**" command on page 127).

Example:

PMCprimo User	Comment
`0.1: ES Prg1`	Sequence Prg1 is defined, which
`002E CH0.1;SV`	prints the value of "**SV**" for test.
`003E NS`	
`0.1:`	
`0.1: ES Prg2`	Sequence Prg2 is defined, which
`002E CH0.1;SA`	prints the value of "**SA**" for test.
`003E NS`	
`0.1:`	
`0.1: DI0.3:1+,Prg1`	Define virtual input 3:1 with sequence Prg1
`0.1: DI0.3:2+,Prg2`	Define virtual input 3:2 with sequence Prg2
`0.1: EI`	Enable all inputs
`0.1: RI0.3`	Show input port 0.3
`0.1:`	
`# Input: 0.3`	
`8 7 6 5 4 3 2 1`	
`0 0 0 0 0 0 0 0`	All inputs are clear.
`E E E E E E E E`	All inputs are enabled.
`0.1:`	
`0.1: BN0.3,11111111`	Every input should inhibit after its change.
`0.1: SO3:1;WT10;CO3:1`	Set and reset of the virtual output makes a
`0.1:`	signal for the virtual input.
`0.1 SV=20000`	Sequenze Prg1 was executed.
`0.1: RI0.3`	Show input port 0.3:
`0.1:`	
`# Input 0.3`	
`8 7 6 5 4 3 2 1`	
`0 0 0 0 0 0 0 0`	
`E E E E E E E B`	The command "**BN**" has effected that input 3:1 was inhibit.
`0.1: SO3:1`	Another input signal has no effect.
`0.1:`	

DE[t.] g:n± Define error output

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8

This command defines the specified output line as an error output. The line is set to the specified state when PMCprimo detects any motor off error condition, and is cleared to the opposite state when the axis is returned to the position control state with the "**PC**" command.

The following error conditions change the state of the error output:

- **Set error (SE command)**
- **Timeout encoder (TO command)**
- **Limit switch detected (DL command)**
- **Motor errors defined with EW command**

Example:

PMCprimo User	Comment
`0.2>  CH1;DE2:1-<CR>`	This command defines output 1 of byte 2 as error output for channel 1
`0.1:  LO2<CR>`	List output definitions byte 2
`# Outputs`	
`0.2:1 - DE1`	Output 1, byte 2 is error output channel 1
`0.2:2   undef.`	
`0.2:3   undef.`	
`0.2:4   undef.`	
`0.2:5   undef.`	
`0.2:6   undef.`	
`0.2:7   undef.`	
`0.2:8   undef.`	
`0.1:  CH1;DE2:1<CR>`	undefine error output 1 of byte 2
`0.1:`	

PO[t.] g:n[±,X1,X2] **Define position trigger output**

Range t:	0 to 59, (depends on number of nodes) or no value	
Range g:	1 to 2, PMCprimo Drive/2, (2 is virtual)	
	1 to 3, PMCprimo 2+2, (3 is virtual)	
	1 to 3, PMCprimo 16+, (3 is virtual)	
Range n:	1 to 8	
Range X1:	±4.000.000	As of version **2.004**: ±2.000.000.000
Range X2:	±4.000.000	As of version **2.004**: ±2.000.000.000

This command defines the specified output line as a position trigger output. If the "**PO**" command is given with a sign, it must be followed by two position values. These define the range of positions, within which the output line goes to the state specified by the sign in the command. A line which has been defined as a position trigger output may be returned to normal operation by entering this command without the sign.

The output and the axis must be from the same node.

Example 1:

PMCprimo User		Comment
`0.2>`	`PO2:5-,500,800<CR>`	This command defines that output 2:5 is low if the position of channel 0.2 is between 500 and 800.
`0.2>`		
`0.2>`	`PO2:5<CR>`	Undefine "**PO**".
`0.2>`		

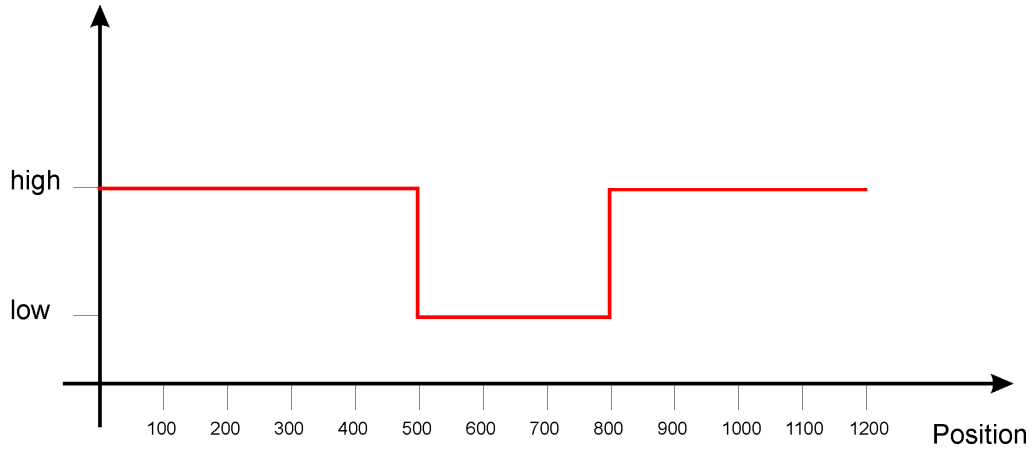


Figure 37: Example 1 for command PO

Example 2:

PMCprimo User		Comment
`0.2>`	`SB1000<CR>`	Set bound to 1000.
`0.2>`	`PO2:5+,0,-200`	Define position output for channel 2
`0.2>`	`PC;VC-`	Move in negative direction.

Unless **version 1.006** it is toggle in the range between –800 and –1000, whereas the bound is set to 1000.

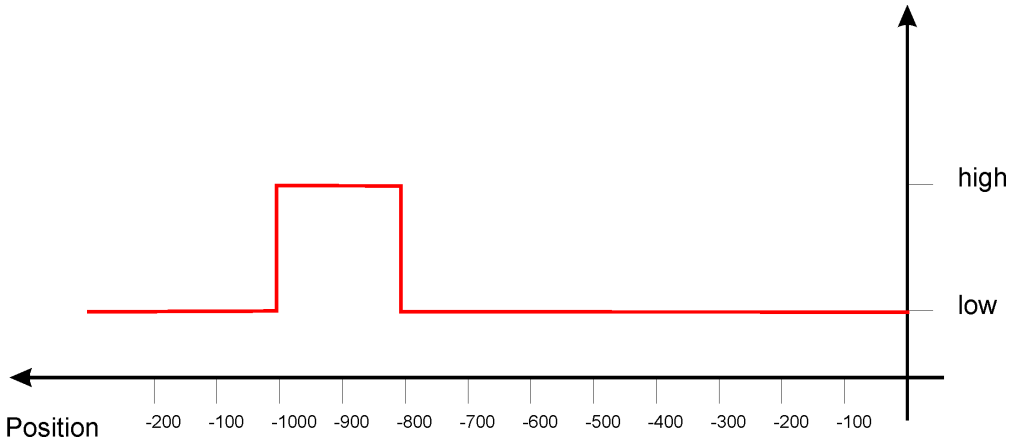


Figure 38: Example 2 for command PO unless version 1.006

Since **version 1.007** it is toggle in the range between 0 and –200, whereas the bound is set to 1000.

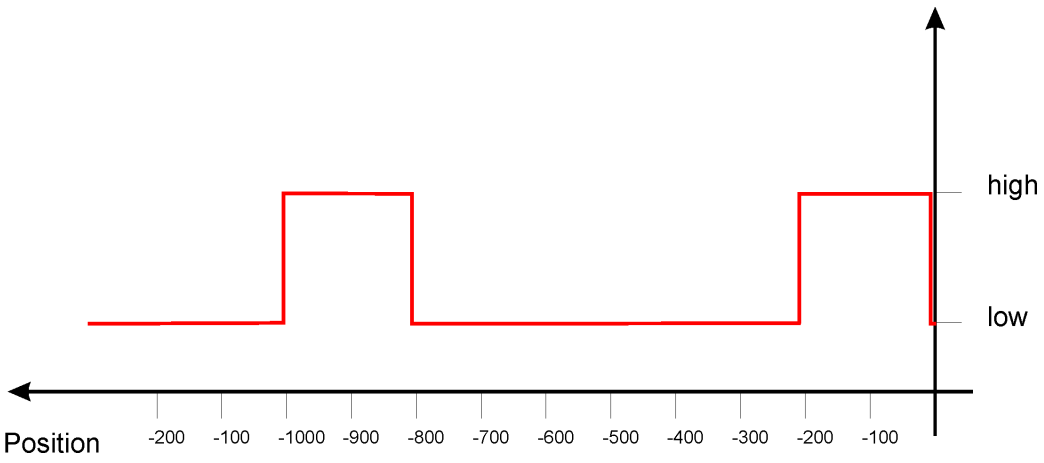


Figure 39: Example 3 for command PO since version 1.007

PHnn Define hysterese for position trigger output

Range nn: 0 to 4.000.000

As of version **2.004** new range for nn: ±2.000.000.000

Enhancement as of **version 1.007.**

The command "**PH**" is used for a hysteresis of the command "**PO**". The hysteresis is added to the switching positions to prevent the position output to switch very fast without moving (for example if the actual position changes 1 increment around the switching position).

VO[t.] g:n[±,X1,X2] Define velocity trigger output

Default t:	0 to 59, (depends on number of nodes) or no value	
Default g:	1 to 2, PMCprimo Drive/2, (2 is virtual)	
	1 to 3, PMCprimo 2+2, (3 is virtual)	
	1 to 3, PMCprimo 16+, (3 is virtual)	
Default n:	1 to 8	
Begin velocity X1:	**±**4.000.000	as of version **2.004**: ±40.000.000
End velocity X2:	**±**4.000.000	as of version **2.004**: ±40.000.000

This command defines the specified output line as an output depending on the velocity. It must be followed by two velocity values. These define the range of velocity, within which the output line goes to the state specified by the sign in the command. A line which has been defined as a velocity trigger output may be returned to normal operation by entering this command without the sign.

The output and the axis must be from the same node.

Example:

PMCprimo User		Comment
0.2>	VO2:5-,500,10000<CR>	This command defines that output 2:5 is low if the
0.2>		velocity of channel 0.2 is between 500 and 10000.
0.2>	VO2:5<CR>	Undefine VO.
0.2>		

VHnn Define hysterese for velocity trigger output

Range nn: 0 to 4.000.000

As of version **2.004** new range for nn: ±2.000.000.000

Enhancement as of **version 1.007.**

The command "**VH**" is used for a hysteresis of the command "**VO**". The hysteresis is added to the switching velocities to prevent the velocity output to switch very fast without moving (for example if the actual velocity changes 1 increment around the switching position).

DBnn Set input debounce time

Range n: 0 to 10000
Default: 5

This command sets up a debounce time for all the digital inputs. It is specified in units of 1 millisecond. The debounce time for reference inputs is set with the "**RH**" command (see page119).

BO[t.] g:n± Define bound overflow output

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8

This command defines the specified output line as a position bound overflow output. Each time the channel passes the position bound set by the "**SB**" command, a logic high or low pulse is output on the specified output line. The sense of the pulse is defined by the sign given in the command. The output pulse lasts for a minimum of 1ms. A line which has been defined as a bound overflow output may be returned to normal operation by entering this command without the sign.

OW[t.] g:n± Define outside window output

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8

Enhancement as of **version 1.008.**

This command defines an output for watching position error. If the position error is bigger than "**SW**" the output is set/ reset. A line which has been defined as an outside window output may be returned to normal operation by entering this command without the sign.

RR[t.] g:n± Define reference reject output

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8

This command defines the specified output line as a reference reject signal output. The "**RR**" output is set to true if any reference error occurs, and is cleared when a valid reference signal is detected. The sense of the output is defined by the sign given in the command. The output state is held until the next valid reference is detected. A line which has been defined as a reference reject output may be returned to normal operation by entering this command without the sign.

A typical application of the "**RR**" command is for a simple product reject facility. If the reference input is triggered by the leading edge of a product on a conveyor belt, detected by a photocell or proximity switch, then the "**RR**" output together with the "**SR**" command indicates when a product is out of position by more than the "**SR**" value. The signal may be used to trigger a product reject actuator if required.

Example:

PMCprimo User	Comment
`0.1> SR500<CR>`	Specified limit for reference error
`0.1> RR1:5+<CR>`	Output 5, byte 1 is reference reject for channel 1
`0.1>`	

If any reference error occurs, then output line 5 is set to high. It stays high until the next valid reference is detected, when it changes to the low state.

If the maximum reference correction (SR) is set to 0, the output state is set always TRUE.

Change as of **Version 2.000**:

If the maximum reference correction (SR) is set to 0, the output state is set always **FALSE**.

AE[t.] g:n± Define analogue limit error output

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8

This command defines the specified output line as an analogue input out of limits error signal. The line is set to the specified state when the analogue input value is outside the high and low limits set by the "**AH**" and "**AL**" commands, and cleared again when the analogue input returns into these limits.

Example:

PMCprimo User	Comment
`0.2>  CH1;AE2:2-<CR>`	This command defines output 2:2 as an analogue error limit output. Output 2:2 is low if the analogue input is outside limits.
`0.1:  LO2<CR>`	List output definitions from byte 2.
`# Outputs 0.2`	
`0.2:1   undef`	
`0.2:2 - AE1`	
`0.2:3   undef`	
`0.2:4   undef`	
`0.2:5   undef`	
`0.2:6   undef`	
`0.2:7   undef`	
`0.2:8   undef`	
`0.1:  CH1;AE2:2<CR>`	A line which has been defined as an analogue error output is returned to normal operation by entering this command without the sign.
`0.1:`	

LI[t.]g List input line definitions

Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 3 (3 is virtual)

This command lists the current definitions of the input lines for the specified group on the display. The list shows the input line number, followed by a sign (+ or -) and a letter representing its function. Lines not defined are left blank. Function inputs also have their command string listed.

Example:

PMCprimo User	Comment
`0.3>  LI1<CR>`	List input line definitions byte 1 Inputs 1 to 8 from byte 1, node 0
`# Inputs 0.1`	
`0.1:1 - DR3`	Falling edge is reference to channel 3
`0.1:2 + PROG1`	Execute sequence PROG1
`0.1:3 - TEST`	Execute sequence TEST
`0.1:4...undef.`	no input line definition
`0.1:5...undef.`	no input line definition
`0.1:6...undef.`	no input line definition
`0.1:7 - DL2`	Limit switch channel 2
`0.1:8...undef`	no input line definition
`0.3>`	

LO[t.]g List output line definitions

Range for t: 0 to 59, (depends on number of nodes) or no value
Range for g: 1 to 2, PMCprimo Drive/2, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)

This command lists the current definitions of the output lines for the specified group on the display. The list shows the output line number, followed by a sign (+ or -) and a letter representing its function. Lines not defined are left blank.

Example:

PMCprimo User	Comment
`0.3>  LO1<CR>`	List output line definitions byte 1
`# Outputs 0.1`	Outputs 1 to 8
`0.1:1  undef`	no output line definition
`0.1:2  undef`	no output line definition
`0.1:3  undef`	no output line definition
`0.1:4  undef`	no output line definition
`0.1:5 - PO3>500<1000`	position output channel 3
`0.1:6 + DE2`	error output: Logic high if motor error occurs
`0.1:7 + BO2`	1ms pulse at bound overflow channel 2
`0.1:8 - AE1`	analogue error output channel 1
`0.3>`	

FSnn Set feedback encoder type

Range n: 0 to 26
Default: 0

This command sets up different encoder feedback options for every axis. The following feedback encoder types can be used:

- Incremental feedback encoder (FS 0-2)
- SSI or Hiperface feedback encoder (FS 5-8, 11-18)
- CANopen feedback encoder (FS 9-10)
- Transmission of feedback encoder values with CANopen for PMCtendo DD4 (FS 19-20)

With transmission of the demand positions with CANopen for PMCtendo DD4, the position control is done by PMCtendo DD4 (FS 21-26).

FS	Option
0	Quadrature x 4
1	Quadrature x 2 (except channel 1 in PMCprimo Drive/2)
2	Quadrature x 1 (except channel 1 in PMCprimo Drive/2)
3	Reserved
4	Reserved
5	Hiperfache/SSI, relative position, binary; 300kHz
6	Hiperfache/SSI, relative position, gray code; 300kHz
7	Hiperfache/SSI, absolute position, binary; 300kHz
8	Hiperfache/SSI, absolute position, graycode; 300kHz
9	CANopen, relative position
10	CANopen, absolute position
11	SSI/Hiperface, relative position with high resolution (as of **version 1.009**)
12	SSI/Hiperface, absolute position with high resolution (as of **version 1.009**)
15	Hiperfache/SSI, relative position, binary; 100kHz
16	Hiperfache/SSI, relative position, gray code; 100kHz
17	Hiperfache/SSI, absolute position, binary; 100kHz
18	Hiperfache/SSI, absolute position, gray code; 100kHz
19	PMCtendo DD4, relative position (chose the PMCtendo DD4 with pd and consider the reference.)
20	PMCtendo DD4, absolute position (chose the PMCtendo DD4 with pd and consider the reference.)

21	CANopen, demand position, relative evaluation, positive reference signal (as of **version 2.000**) (chose the PMCtendo DD4 with pd consider the reference.)
22	CANopen, demand position, relative evaluation, negative reference signal (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)
23	CANopen, demand position, absolute evaluation, offset set with command RF (page 112) (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)
24	CANopen, demand position, relative evaluation, positive reference signal. The actual following error and the actual current is received. (as of **version 2.004**) (chose the PMCtendo DD4 with pd and consider the reference.)
25	CANopen, demand position, relative evaluation, negative reference signal The actual following error and the actual current is received. (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)
26	CANopen, demand position, absolute evaluation, offset set with command RF (page 112). The actual following error and the actual current is received. (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)

With a CAN-Baudrate of 1 Mbit/s and a cycletime of 1ms do “syncsrc 3” and “fpga 3” with the chosen PMCtendo DD4.

As of **version 1.009**:

FS 11 and 12, Hiperface relative and absolute:

In comparison with FS5,6 or 7,8 a higher resolution for the interchange of the demand values is used.

Standard: Interchange of demand value in1 1/min.
With FS11 and 12 follow resolution is set:
NB28 (28 bit). 1/140 1/min (maximal resolution PMCtendo DD4)
NB27 1/70 1/min
NB26 1/35 1/min
NB25 1/ 17,5
NB24 1/8,75 1/min
NB23 and less 1 1/min

The resolution was selected in this way, with it the values of “**KF**” (see page 100) are in the range of 0 to 65535. With “**NB**” of 24 to 28 the “**KF**” and “**KP**” value (see page 99) is always the same because of the division with 2. The optimum “**KF**” value in this case is 32812. With “**NB23**” and less the “**KF**” and “**KP**” value must adjust as normal.

As of **version 2.000**:

This extension operates only with PMCtendo DD4 **version 4.94** and higher.

FS 21, 22 and 23 demand position with CANBus. The position control is made by PMCtendo DD4. The position control of PMCprimo (KP, KF ect.) is not in use. With “**FS21**” and “FS**22**” the latch function of PMCtendo DD4 can be activate. Therefore the drive command “IN2MODE” must set to 26 and reference input is input 2 of PMCtendo DD4. (SAVE+COLDSTART must made)

The solution can set with the NB command (page 159) (NB24 means 4096 increments per revolution. The rolling direction can be set with bit 5 of “**CW**” command (page 48).

With FS21 and 22 the referencing is supported completely (IN, RM, RW, FW, FH, FL, ZH, ZL). There must set no “**DZ**” or “**DR**”. With FS21 or 22 no reference input of PMCprimo is allocated but referencing can be made.

With every “**PC**” command the actual position of the drive is read and the position counter of PMCprimo is set to it. If the motor is moved manhandled the change is seen after the next “**PC**” command. Therefore the position is not lost. After first switch on it isn’t made by relative position. The actual position in PMCprimo is still present. The behavior is the same like an incremental encoder.

As of **version 2.004**:

This extension operates only with PMCtendo DD4 **version 5.16** and higher.

New settings “FS24” to 26. As with “FS21” to 23 the demand position is sent to PMCtendo DD4 and the internal position loop of the PMCtendo DD4 is used. The referencing with “INMODE2 26” is also available. The difference is that the actual following error and the actual current is received.

To get the right value the drive commands “PGEARI” and “PGEARO “are changed automatically from PMCprimo to 1048576 (if “PRBASE” is 20) or 65536 (if “PRBASE” is 16). The setting of the following error (“PEMAX”) inside PMCtendo DD4 has to be adjusted by the customer.

The function should be only for service or adjusting. If the machine is finished then the “**FS**” should be set to “FS21” to 23 to decrease the bus load

Overview FS settings and technical data:

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
0	Incremental encoder X4 multiplication PMCprimo Drive channel x.1: 4096 increments/turn	PMCprimo Drive Channel x.1: At the end of the 1ms Interrupt. Depending of processor load 200-300us Analogue output: In the 1ms interrupt after calculation of the position loop. Depending of processor load 200-300us	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
1	Incremental encoder X2 multiplication PMCprimo Drive channel x.1: 2048 increments/turn	see FS0	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
2	Incremental encoder X1 multiplication PMCprimo Drive channel x.1: 1024increments/turn	see FS0	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
5	SSI encoder : relative position, binary code PMCprimo Drive channel x.1 : Hiperface resolution with NB	see FS0	If PMCprimo Drive channel x.1 Hardware register < 1us With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
	changeable relative position						
6	SSI encoder : relative position, Gray code PMCprimo Drive channel x.1 : Hiperface resolution with NB changeable relative position	see FS0	If mcD channel x.1 Hardware register < 1us With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
7	SSI encoder : absolute position, binary code PMCprimo Drive channel x.1 : Hiperface resolution with NB changeable absolute position	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
8	SSI encoder : absolute position, Gray code PMCprimo Drive channel x.1 : Hiperface resolution with NB changeable absolute position	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
9	CAN open encoder relative position	see FS0	Depending for CAN cycle time 1,2 or 4ms	1	Yes, with PMCprimo Drive and channel x.1	Yes	No
10	CAN open encoder absolute position	see FS0	Referencing not possible	1	Yes, with PMCprimo Drive and channel x.1	Yes	No
11	PMCprimo Drive channel x.1 resolution with NB changeable, resolution of demand speed with DPRam depends from NB setting, relative position	see FS0	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
12	PMCprimo Drive channel x.1 resolution with NB changeable, resolution of demand speed with DPRam depends from NB setting, absolute position	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
15	see FS 5, frequency 100kHz	see FS0	If PMCprimo Drive channel x.1 Not available With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
16	see FS 6, frequency 100kHz	see FS0	If PMCprimo Drive channel x.1 Not available With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
17	see FS 7, frequency 100kHz	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
18	see FS 8, frequency 100kHz	Depending for CAN cycle time. Transmission with the next synchronisation message. 1,2 or 4ms	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
19	PMCtendo DD4 demand velocity and actual position. Relative position	see FS18	Depending for CAN cycle time 1,2 or 4ms	2	Yes, with version 3.56a or 4.94a	Yes	No
20	PMCtendo DD4 demand velocity and actual position. Absolute position	see FS18	Referencing not possible	2	Yes, with version 3.56a or 4.94a	Yes	No
21	PMCtendo DD4 demand position Relative position and positive reference signal	see FS18	Hardware register PMCtendo DD4 < 1us	1	No	No	Yes
22	PMCtendo DD4 demand position Relative position and negative reference signal	see FS18	Hardware register PMCtendo DD4 < 1us	1	No	No	Yes
23	PMCtendo DD4 demand position Absolute position	see FS18	Referencing not possible	1	Nein	No	Yes
24	See FS21 but with following error and	see FS18	Hardware register PMCtendo DD4 < 1us	2	Yes, with version 5.x	Yes	Yes

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
	actual current						
25	See FS22 but with following error and actual current	see FS18	Hardware register PMCtendo DD4 < 1us	2	Yes, with version 5.x	Yes	Yes
26	See FS23 but with following error and actual current	see FS18	Referencing not possible	2	Yes, with version 5.x	Yes	Yes

Example 1:

Requirement: PMCprimo Drive with incremental feedback encoder (5000 impulses)

Adjustment FS:
0: quadruple impulse analysis => 20000 increments / rotation
1: twofold impulse analysis => 10000 increments / rotation
2: single impulse analysis => 5000 increments / rotation

Example 2:

Requirement: PMCprimo Drive with Hiperface feedback encoder

The resolut is set with the command NB (page 159).

Adjustment FS:
5 or 6: The position is analysed relatively. The SB value (page 108) is relevant and should be a multible of 2^n, to obtain the position after a restart.

7 or 8: The position is analysed absolute. The SB value (page 108) is not relevant. The via NB adjusted value determines the cycle length.
1: twofold impulse analysis => 10000 increments / rotation
2: single impulse analysis => 5000 increments / rotation

Example 3:

Requirement: PMCprimo 16+ with PMCtendo DD4 and Hiperface feedback encoder networked with CANopen.

The desired drive must be choosen with the PD (page 190) command.
The resolution is set with the NB (page 159) command.

Adjustment FS:
19: The position is analysed relatively. The SB value (page 108) is relevant and should be a multiple of 2^n, to obtain the position after a restart.
20: The position is analysed absolute. The SB value (page 108) is not relevant. The via NB adjusted value determines the cycle length.

NBnn Set number of bits for SSI encoder

Range: 12 to 32
Default: 24

This command sets the number of data bits used when the channel is configured for use with an SSI encoder. It allows several different types of SSI encoder to be used in different applications, from single turn 12 bit to multiturn 25 bit models. The number of data bits may be set differently on each channel if required.

Hiperface:

Range: 12 to 28

For Hiperface encoder the last 12 bit of the NB value are always reserved for the number of rotations. The remaining bit defines the resolution for one rotation. For example NB28 the motor moves 65.537 increments per resolution.

FCnn Feedback change encoder

Range for nn: 0 to 2 (PMCprimo Drive/2)
0 to 3 (PMCprimo 2+2)
1 to 16 (PMCprimo 16+) adjustable with command “**CD**”
Default: 0

With this command it is possible to change the encoder feedback from axis 'nn' to the actual axis. For example axis 1 of PMCprimo Drive should connect with an incremental encoder, its only possible by using the “**FC**” command:

```
CH1; FC2
```

An encoder is for example a possible application, which is fitted after a gearbox and shall be used as actual value source for the motor. The encoder is connected at the free encoder input and is allocated to the axis 2 with PMCprimo Drive. Axis 1 (motor axis) can take over the positions of the axis 2 with this function and, therefore, the position control is calculated by this desired value.

A reference input, “**FS**” or “**DZ**” must be defined at the channel the “**FC**” command refers.

MSnn Master scale factor

Range for nn: 0 to 8
Default: 0

With this command it is possible to scale a master position. The master position is multiplied with the factor sn. This makes only sense if the master position is averaged with the command “**PT**” (page 47), or else the position changes are to big.

This possibility gives the “**PT**” command a higher resolution of the measuring system. This is needed, when a slave axis must follow the master axis with a higher transmission ratio. With a transmission ratio of 10:1, the slave must drive 10 increments when the master drives one increment. This is noticeable at the motor by noises and higher temperature development. Will the position now scaled and averaged, the transmission ratio will be scaled down and the effect will be reduced.

Enhancement as of **version 2.002**:
It is possible to make an assignment to a variable.

8.12 Display Commands

Display of datas, like position, velocity, position error.

DP Display actual position

Displays current actual position in encoder counts.

As of **version 2.006**:
With the settings FS21 to 26 (page 151) now the actual position is read from the drive. If the motor is turned by hand (if motor is switched off) then until now the actual position was not shown because with this settings the actual position is not send. Now the position is read with DP and the real actual position is shown.
Note: The command DM (scope) is not changed. The improvement is done only for the command DM.

DD Display demand position

Displays current demand position in encoder counts.

DV Display velocity

Displays the current measured velocity of the channel in encoder counts per second. If speed averaging is enabled by the "**VT**" command (see page 172), then the displayed velocity value is the average measured velocity with the time base of "**VT**".

FE Display following error

Displays the closed loop following error in encoder counts. This is provided for applications which need to read the following error into a variable.

DF Display reference error

Displays measured absolute position error relative to the last valid reference input, in encoder counts. For more details see section for reference commands on page 103.

DG Display actual reference error

Command as of **version 2.000**:

Displays the position error which is really adjust in increments. The command "**DF**" whereas shows the measured error.

For more details see section for reference commands on page 103.

DS Display snapshot position data

Displays the last absolute position measured when a snapshot input signal was detected. For more details see the "**PS**" configuration command on page 137.

DA Display analogue input

Displays the current channel's analogue input signal voltage as a number in the range ±2047, corresponding to ±10V.

DT Display time

Displays current time, in hh:mm:ss format.

The time starts after switching on with 00:00:00. Its not possible to set the actual time.

DO Display mode off

Turns off the "**DM**" continuous display. This command may also be used to turn off the trace mode "**TR**" display.

GM Display mapped axis bound position

Command as of **version 1.006.**

This command displays the actual bound. If bit 7 of command "**MW**" is set (page 81) the bound of the slave is automatic set to the map and the actual scalemap. The calculated value can be displayed with the command "**GM**". If mapping is not active, the "**SB**" value (page 108) is displayed.

GW Get wraparound offset value

Command as of **version 1.006.**

"**GW**" displays the offset between the actual slave position und from the map calculated position. An offset with the command "**MB**" and "**MF**" (page 77) is not considered. If no drift is allowed, the "**GW**" value should not change. The automatic cyclic length correction may help here (command "**MW**" Bit 7 page 81).

DM[nn] Continuous display mode

Range nn: 1 bis 65.535, or no value

If no parameter value is given, this command turns on a continuous display of demand position, measured position, position error, and time. The time step is constant and depends on the baud rate of the serial interface (for PMCprimo it's 9,6 kbaud). The parameter 'T' displays a timer counter. The difference of two successive values is the actual time step.

If a parameter value is given, PMCprimo prints that number of lines of the position data, recorded at the full time step of 1 millisecond.

Example 1:

PMCprimo User	Comment
`0.2> DM2<CR>`	display 2 lines
`0.2 D=0000918 P=0000918 E=+0000000 T=0000001`	
`0.2 D=0000920 P=0000920 E=+0000000 T=0000002`	
`0.2>`	

Following values are displayed:

D: demand position
P: actual position
E: position error
T: timer counter in ms

Example 2:

PMCprimo User	Comment
`0.2> DM<CR>`	The display mode at channel 2 is active until the command DO stops it.
`0.2 D=0000918 P=0000918 E=+0000000 T=00082231`	
`0.2 D=0000918 P=0000918 E=+0000000 T=00082303`	
`0.2 D=0000918 P=0000918 E=+0000000 T=00082376`	
`0.2 D=0000918 P=0000918 E=+0000000 T=00082449DO<CR>`	
`0.2>`	

TR[nn] Enable trace mode

Range nn: 0 to 10.000

This command controls a continuous trace display. It allows various data values to be displayed via the serial port, similar to the “**DM**” command, but it allows data from more than one channel to be simultaneously displayed, and it supports a wide range of data values.

The data values to be traced are specified by the “**TW**” command. It is possible to display 4 values defined with the command “**TW**” (see page 165). The “**TR**” command with parameter log 'nn' values in a time step defined by “**TT**” (see page 164). It's possible to define a trigger with the command “**TH**” (see page 167).

If no parameter value is given, the sample time is constant and depends on the baud rate of the serial interface (for PMCprimo it's 9,6 kbaud).

Enhancement as of **version 1.008:**
With “**TR0**” the last recorded datas can be print again.

Example 1:

PMCprimo User	Comment
`0.1V TH0`	Deactivate the trigger function
`0.1V TT4`	time step set to 4 ms.
`0.1V TW1,1`	channel 1 should log demand positions
`0.1V TR2`	Start recording of 2 values
`0.1V`	
`D0.1=00000123 T=00000004`	demand position is 123, T shows the elapsed time.
`D0.1=00000125 T=00000008`	
`0.1V`	

TT[nn] Time step to trace mode

Range: 0 to 10.000

This command defines a time step for the trace mode TR.

Enhancement as of **version 2.002**:
It is possible to make an assignment to a variable.

TW[n,m,VarName] **Configuration word for trace mode**

Range n: 1 to 4
Range m: 0 to 32
Range VarName: Variable name for type 43 and 44 (as of **version 2.000**)

It can choose between four record channel. With 'm' the data value to be traced is specified.

Enhancement as of **version 2.000**:
For command type 43 and 44 the variable name must indicate. The selected variable is getting automatic global, because it is logged in the channel task. The settings TWn.43,$Name and TWn,44,$Bn are not saved but listed with "**LA**".

Nr. Type	Data value	Prefix letter
m=0:	No Value	
m=1:	Demand position	D
m=2:	Actual position	P
m=3:	Position error	E
m=4:	Demand velocity	V
m=5:	Actual velocity	W
m=6:	Average actual velocity	X
m=7:	Reference error	R
m=8:	Snapshot position	S
m=9:	Change in demand	U
m=10:	Change in position	C
m=11:	Analogue input value	A
m=12:	Analogue loop error	L
m=13:	Actual scale map	U
m=14:	Master axis position	MP
m=15:	Master axis velocity	MV
m=16:	Position differential	DP
m=17:	Velocity differential	DV
m=18:	Actual current drive	IS
m=21:	Input byte 1	I1
m=20	Change in demand absolute value (since **version 2.004**)	UB
m=22:	Input byte 2	I2
m=31:	Output byte 1	O1
m=32:	Output byte 2	O2
m=40	Motor error (since **version 2.000)**	ME

Nr. Type	Data value	Prefix letter
m=41	Error variable (since **version 2.000)**	$F
m=42	Status variable (since **version 2.000)**	$S
m=43	Busvariable (since **version 2.000)**	$Bn
m=44	Variable (since **version 2.000)**	$Name
m=61	CAN input byte 1 (since **version 2.000)**	I1
m=62	CAN input byte 2 (since **version 2.000)**	I2
m=63	CAN input byte 3 (since **version 2.000)**	I3
:	:	:
:	:	:
m=71	CAN output byte 1 (since **version 2.000)**	O1
m=72	CAN output byte 2 (since **version 2.000)**	O2
:	:	:
:	:	:

THnn Trigger channel for trace mode

Range nn: 0 to 4

It's possible to trigger on the defined channel 'nn' of “**TW**”. The trigger level is set by “**TL**” and the trigger transition is set by “**TF**”.

TLnn Trigger level for trace mode

Range nn: ±2.000.000.000

This command defines the trigger level for the trace mode “**TR**”. If the value of the trigger channel “**TH**” is equal to “**TL**” and the transition is also equal to “**TF**”, the trace mode starts.

Enhancement as of **version 2.002**:
It is possible to make an assignment to a variable.

TFnn Trigger transition for trace mode

Range nn: 0 or 1, as of **version 2.000**: 2

This command defines the trigger transition for “**TR**”. If the value of the trigger channel “**TH**” is equal to “**TL**” and the transition is also equal to “**TF**”, the trace mode starts.

Enhancement as of **version 2.000**:
Trigger with variable:
0: If the value of variable changes and the value is lower than “**TL**” the trace mode starts.

1: If the value of variable changes and the value is higher than “**TL**” the trace mode starts.

2: If the value of variable changes and is equal to “**TL**” the trace mode starts.

With other records:
0: Trigger with positive edge
1: Trigger with negative edge
2: Trigger with both edges

Enhancement as of **version 2.002**:
It is possible to make an assignment to a variable.

TPnn Pretrigger for trace mode

Range nn: 0 to 10.000

If “**TR**” is used with trigger “**TH**”, it is possible to define a pretrigger with “**TP**”. In this case, the datas are stored permanent in a ring buffer, in order the case history of the last 'nn' milliseconds after trigger is registrated.

TD[mm][,nn] Display last value of trace mode

Range nn: 1 to 4 channel no.
1 to 10.000 output line

This command displays the last value logged by the "**TR**" command. Its possible to make an assignment to a variable.

Enhancement as of **version 1.008:**
With the command "**TD**" without any parameter the last record of '**TR**' can print again. With parameter 'nn' and 'mm' you can print or set to a variable every value recorded by '**TR**'. The command '**TT**' and '**TW**' must not be changed after recording with '**TR**'.

?? Print help display

This command prints a menu list:

```
Help - Main menu
=================
 ?1 -->  Command summary
 ?2 -->  Error and status codes on display
 ?3 -->  Used abbraviations
 ?4 -->  Operators
 ?5 -->  Variables
 ?6 -->  Programs and maps
 ?7 -->  Map generator
 ?8 -->  Values of status variables
```

With enter of '?n' you can select the special help..

For every command it can ask for help by appending a '?' to the command.

Example:

PMCprimo User	Comment
`0.2>  DP?<CR>`	Ask for a help about the "**DP**" command
`DP:  Display measured position`	
`Syntax:  DP`	
`Example: DP, $Variable=DP`	
`0.2>`	

LE Display last error

This command redisplays the last 10 error messages for the last error detected by PMCprimo. It is useful for finding an error message which has stopped PMCprimo when there is not normally a display connected to the machine.

LGnn Set language

Range n: 0 to 3
Default: 0

This command allows the user to choose the language for the messages which are given from PMCprimo to the user. The following languages are available at the moment.

LG	Language
0	German
1	English
2	Italy
3	French

SYb Set system prompts

Range n: 0 or 1
Default: 0

Setting "**SY**" to 1 inhibits the normal echo of characters sent to the terminal. The error messages are displayed only with there error numbers.

8.13 Phase Advance

PA[t.] g:n,X Set phase advance scale factor

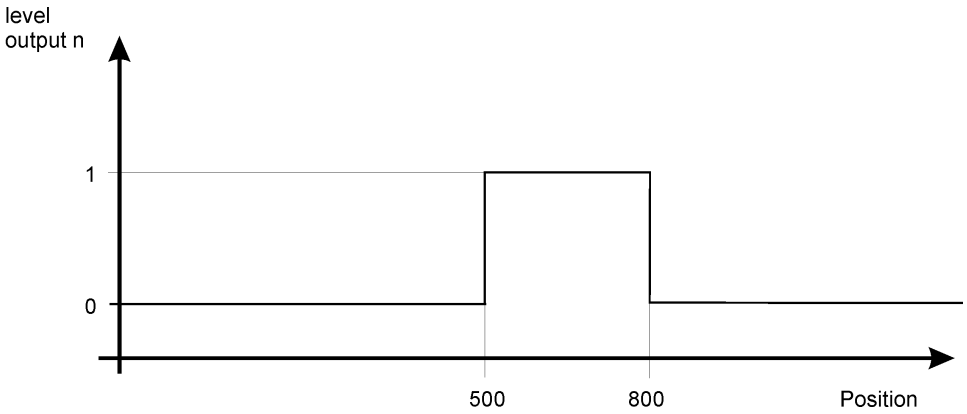
Range t: 0 to 59, (depends on number of nodes) or no value
Range g: 1 to 2, PMCprimo Drive, (2 is virtual)
1 to 3, PMCprimo 2+2, (3 is virtual)
1 to 3, PMCprimo 16+, (3 is virtual)
Range n: 1 to 8
Range X: 0 to 65.535
Default: 0

The phase advance feature is mechanism for shifting all position trigger output signals programmed on the current axis by some amount dependent on the instantaneous measured speed. The definition belongs to the output t.g:n. The phase advance is defined as a shift proportional to the current speed of the motor. This command sets the scale factor between the measured speed and the actual phase advance. The scaling of the phase advance is given by the expression

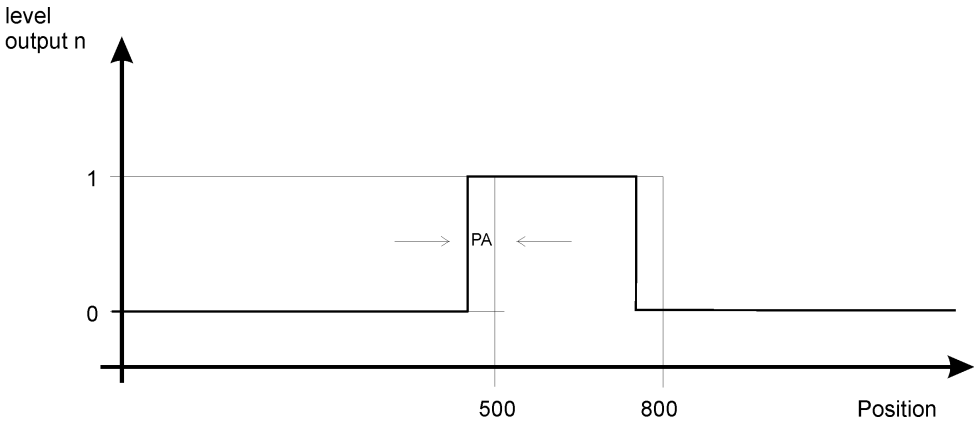
$$\text{phase advance} = (\text{speed}/256) \times (\text{PA}/256)$$

where the axis speed is measured in encoder counts per second. For example, with a measured speed of 20,000 counts per second, a value for "**PA**" of 500 gives a phase advance of 153 encoder counts.

Position output with PA0:



Position output with phase advance at velocity = 1000 counts/second and phase advance PA = 2560:



Position output with phase advance at velocity = 5000 counts/second and phase advance PA = 2560:

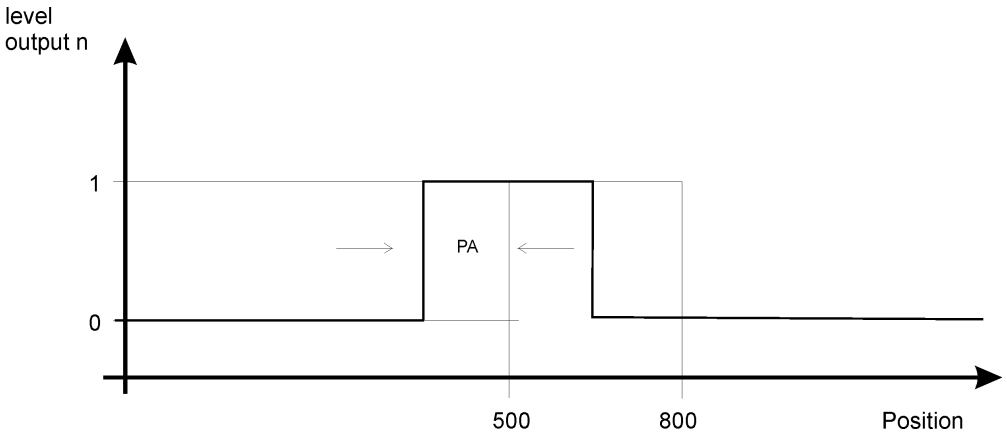


Figure 40: Phase advance

VTn Set velocity averaging time constant

Range n: 1 to 10000 milliseconds
Default: 1

When using the phase advance facility, the measured axis speed is used to calculate the required amount of phase advance. The measured speed is calculated from successive encoder positions at 1ms intervals, and so the measured speed is only accurate 100 encoder counts per second. When used with the phase advance, any variation in the measured speed causes a varying phase advance term, giving erratic operation of any advanced output signals. In this case speed averaging is required to maintain correct operation. If n is equal to 1, PMCprimo calculates no averaging time constant.

The averaged speed value is returned by the "**DV**" display velocity command.

BAn Set map base advance

Range n: 0 to 65.535
Default: 0

The map base advance is a mechanism for shifting the mapped position of a slave axis, relative to the master axis, by some amount dependent on the current master axis speed. The base advance is applied to the slave axis in the same way as the fixed "**MB**" map base offset value, and thus is defined as a shift along the master axis proportional to master axis speed ("**MB**" command see page 77). This command sets the scale factor between the measured master axis speed and the actual map base advance. The scaling of the base advance is given by the expression

$$\text{map base advance} = \frac{\text{master speed}}{256} \times \frac{BA}{256}$$

where the master speed is measured in encoder counts per second. For example, with a measured master axis speed of 10.000 counts per second, a value for "**BA**" of 2000 gives a map base advance of 30 encoder counts.

The average speed of the master axis can acquired with the "**BT**" command.

This factor can also regarded as adjusted time. The value 1 conforms 15,625 microseconds. A phase shit of 64 conforms 1 ms and thus the way which the master puts back in this time.

BTn Set base advance time constant

Range n: 1 to 10000 milliseconds
Default: 1

When using the map base advance facility, the master axis speed is used to calculate the required amount of advance. When the master axis is also controlled by PMCprimo, the demand speed for the master axis is available to the slave, giving very smooth operation. The "**BT**" command sets up an averaging mechanism on the slave axis. If n is set to 1, PMCprimo calculates no averaging time constant.

The averaged velocity is also used by the "**XM**" command (see page 74) for recognizing if the Master is moving.

Enhancement as of **version 2.002**:

The master velocity can be averaged in speed mapping with the command "**BT**". Damping of rapid movements of the master is possible with it.

8.14 Analogue Control

This section describes facilities for performing analogue loop control, used for tension control applications. Closed loop tension control is performed by calculating the required speed ratio between a master and slave axis in mapping, with feedback on the analogue input.

The tension control loop reads the analogue input at the normal system sample rate, and compares it with the analogue setpoint value. The difference between the setpoint and the input is the analogue loop error, and this is used in a closed loop control algorithm to determine the required action.

The ratio control algorithm used is of the following form:

$$i = SM + SM \times [AP\ e_i + AI\ \Sigma e_i + AD\ (e_i - e_{i-1})]$$

i	=	ratio
SM	=	default ratio (SM-command)
AP	=	proportional gain constant
AI	=	integral gain constant
AD	=	differential gain constant
e_i	=	tension error (= tension setpoint – measured tension)

In closed loop analogue control, the dynamic behaviour of the system depends on these gain constants, and on the mechanical characteristics of the system being controlled. Tuning the control system to get best performance on a particular mechanical setup requires setting up these gain commands.

AMn Set analogue control mode

Range n: 0 to 1
Default: 0

This command selects the analogue closed loop control mode. “**AM1**” enables tension control mode, and “**AM0**” disables the analogue control functions. The analogue control loop may be enabled and disabled at any time. The “**AM**” value is not saved.

Setting “**AM1**” also sets bit 0, 1 and 4 of “**MW**” to 1. This forces the use of the software clutch and speed mapping, as required for tension control.

Activate mapping with the command “**XM**” sets bit 0, 1 and 4 if “**MW**” to 1. This forces the use of the software clutch and speed mapping, as required for tension control. If either of “**AW**” bits 4 and 5 are also set to 1, then the unit automatically initialises the map scale factor before executing the map under tension control. The system measures the distance moved by the master and/or slave axes between the two positions where the analogue input high and low limits are exceeded, and stores these in the “**AR**” parameters. The ratio of these two values gives a good estimate of the initial map ratio required by the tension control loop and is used at the start of synchronization. This allows the system to reach its correct steady state ratio much more quickly than if it starts from the default “**SM**” value, particularly if the system is not at its normal starting position.

An example where this is useful are winding or unwinding applications. In normal circumstance the system always starts with either full or empty spool of material, and the initial scale factor is set in the “**SM**” parameter. Restarting the machine with the same spools uses the scale factor last calculated when the machine was stopped, and the tension loop restarts smoothly. However, if the machine may be started with spools that are partly filled to an unknown diameter, then the normal “**SM**” value is not at the correct value of running at the new spool diameter. The analogue range initialisation function allows the required ratio with the actual spool diameters to be measured. The machine may be started with the new spool without any large transients while the tension control loop stabilizes from the initial default “**SM**” value to the new scale factor.

APnn Set analogue control proportional gain

Range nn: 0 to 65535
Default: 10.000

This command sets the proportional gain of the tension ratio control loop. The proportional gain acts on the measured analogue error, which is calculated as the difference between the required set point and the value measured by the analogue input. High gain gives the system a faster response and tighter control, but if the gain is too high the system may oscillate. For best results, the proportional gain as low as possible to avoid overshoot or oscillation, while still achieving the required control response.

AInn Set analogue integral control gain

Range nn: 0 to 65535
Default: 0

This command sets the gain for the integral term in the tension ratio control loop. When integral control is used, the system integrates the analogue error by adding the current error to a running total. Integral gain is useful to allow for long term changes in ratio, as required for tension control in winding/unwinding applications.

ADnn Set analogue differential control gain

Range nn: 0 to 65535
Default: 0

This command sets the gain of the differential term in the tension ratio control loop. This term uses the differential of the analogue error (rate of change of error), which represents the velocity error of the system. This is useful where the analogue error is changing rapidly, and provides damping in the analogue control loop.

ACnn Set analogue control set point

Range n: ±2047
Default: 0

This command sets the set point for the analogue control loop. The loop error is calculated by subtracting the measured analogue input value from the set point. It may be incremented and decremented by the "**IP**" command (see page 47) when IS is set to 2.

AWbb Set analogue control options word

Range bb: 0 to 1111.1111
Default: 0

Enhancement as of version **2.001**:
It is possible to set every bit with the wild card 'X'. It effects that the value of this bit will not be changed.

This command allows the user to modify the operation of the analogue control loop in various ways. The value is entered as a binary number, with each bit controlling a different aspect of the function. Leading zeros may be omitted. The bit functions for the analogue control options word are described below (Bit 0 is the right and bit 7 the left one).

Bit 0: This bit controls the behaviour of the system when mapping is started with the "**XM**" command and tension control is enabled by setting "**AM1**".

- 0: When set to 0, the slave axis goes immediately into mapping, using the software clutch, with the map ratio determined by the tension control loop.
- 1: When set to 1, the slave axis moves at jog speed (set by "**SS**") until the measured analogue input reaches the tension control set point, and then drops into mapping. This is intended to initialise tension control when the master axis is stopped.

Bit 1: Not in use

Bit 2: This bit controls the analogue control integral term. It allows the analogue control integral term to be turned off if required, for example if the controlled motor is disabled. This prevents the system from integrating up any static analogue error and giving a large transient when the motor starts again.

- 0: The analogue control integral term is disabled
- 1: The analogue control integral term is enabled

Bit 3: This bit sets the initial ratio when the slave axis is mapped to the master axis and tension control is enabled.

- 0: The initial ratio is the "**SM**" value
- 1: The ratio value previously is kept. Normally this is the last value calculated by the tension control loop before the slave axis was stopped.

Bit 4: This bit controls the automatic measurement of the distance between the high and low analogue limits (the analogue range distance) on the slave axis when the "**XM**" command is executed. Refer to the "**AR**" and "**XR**" commands for more details.

- 0: The slave axis analogue range initialisation is disabled.
- 1: The slave axis analogue range initialisation is enabled.

Bit 5: This bit controls the automatic measurement of the distance between the high and low analogue limits (the analogue range distance) on the master axis when the "**XM**" command is executed. Refer to the "**AR**" and "**XR**" commands for more details.

- 0: The master axis analogue range initialisation is enabled.
- 1: The master axis analogue range initialisation is disabled.

Bit 6: This bit sets the direction of the tension takeup move when the slave axis is started with tension control enabled, and "**AW**" bit 0 set to 1.

0: The direction of the takeup move is the same as the sign of the initial tension error.

1: The direction of the takeup move is the opposite as the sign of the initial tension error.

Bit 7: This bit determines the sense of the analogue control loop.

0: An increase in the analogue error gives an increase in the output value.

1: An increase in the analogue error gives a decrease in the output value.

ATnn Define analogue input type

Range nn: 0 to 2
Default: 0

It's possible to choose between different input types for tension control:

AT0: Analogue input is "**DA**" value

AT1: Reference error is "**DA**" value

AT2: Motor current is "**DA**" value

Enhancement as of **version 2.000**:
AT3: Analogue input is "**DG**" value (see page 161)

ARnn Define analogue range distance

Range nn: 0 to 65535
Default: 256

This command defines or displays the distance between the analogue input high and low limits on the current channel, called the analogue range distance. It may be used to set this value for an axis that is not driven by the unit, or to display the value measured by the "**XR**" initialisation function.

MMnn Define master axis analogue range distance

Range nn: 0 bis 65535
Default: 256

This command defines or displays the distance moved by the current channel's master axis between the analogue input high and low limits, called the master analogue distance. It may be used to set this value for a master axis that is not driven by the unit, or to display the value measured by the "**XR**" initialisation function.

CR Calculate initial ratio from analogue range distances

CR calculates the ratio between the “**AR**” and “**MM**” values on the master and slave axes, and uses the result as the map scale factor. The ratio is calculated from values “**AR**” and “**MM**” as **AR**(slave)/**MM**(master), in much the same way as the “**BR**” command does for the bounds values.

If automatic measurement of the analogue range distance is enabled on either master or slave axes, the “**XM**” command performs an automatic ratio calculation using “**CR**”.

XRn Execute analogue range distance initialisation

Range n: 0: initialises slave axis
1: initialises master axis

“**XR**” executes the analogue range initialisation function manually on the current axis. The measured values are stored in the “**AR**” (XR0) or “**MM**” (XR1) parameter. The command is always used on the slave channel. Before initialising the master with “**XR1**” the command “**ML**” must be done.

The analogue range initialisation function does the following:

The motor moves in one direction, chosen so as to move initially towards the set point, until one analogue input limit is crossed, stores the current position and stops. It then reverses and moves until the other analogue limit is crossed, stores this second position, and saves the difference between the two positions as the “**AR**” (slave with XR0) or “**MM**” (master with XR1) value.

Setting “**AW**” bits 4 and/or 5 on the tension controlled slave axis enables automatic measurement of the analogue range distance on the slave and/or master channel respectively at the start of mapping.

AHnn Set analogue input high limit

Range n: ±31bit value
Default: +2047

This command sets the high limit for the analogue input. If the analogue input value exceeds this value, then the "analogue input high limit exceeded" error message is displayed. If bit 3 of the error options word “**EW**” is set to 1, then this is a motor error, and the axis shuts down to the motor off state.

ALnn Set analogue input low limit

Range n: ±31bit value
Default: -2047

This command sets the low limit for the analogue input. If the analogue input value exceeds this value, then the "analogue input low limit exceeded" error message is displayed. If bit 3 of the error options word “**EW**” is set to 1, then this is a motor error, and the axis shuts down to the motor off state.

8.15 Variable commands

The variable database is a centralized facility which is accessible to all tasks in the system and holds a set of integer variables. Because variables are generally accessible, it is possible for the user to change a variable using an operator's panel and for the variable to be used subsequently to set a motor parameter in PMCprimo. Similarly a variable can be se to some motor parameter, such as the position, which can then be displayed on an operator's panel. A variable can also be set up to trigger execution of a command string on PMCprimo. This means that a button of an operator's panel can be set to update a variable which in turn triggers an action on PMCprimo.

A name is defined with the following rules:

- A variable name consists of up to 20 characters which must be numbers '0-9' or letters 'A-Z', '_' or '.'.
- Upper and lower case letters are equivalent. For example 'POS' and 'pos' refer to the same variable.
- Every variable name begins with a '$'-sign.

It is possible to set a value to a variable hexadecimal. Therefore the prefix '0x' must be written.

Example:

PMCprimo User	Comment
`0.2> $SPD=5000<CR>`	Set the value 5000 to the variable $SPD.
`0.2> $SPD=0x1388<CR>`	Also set the value 5000 to the variable $SPD.

Variables can be used instead of the numeric values denoted with n, nn or m.

Example:

PMCprimo User	Comment
`0.2> SV$SPD<CR>`	This command sets the velocity to the value of $SPD:
`0.2>`	

Converse a value of a parameter can assigned to a variable.

Example:

PMCprimo User	Comment
`0.2> $SPD=SV<CR>`	
`0.2>`	

A variable can be defined as a trigger variable so that when it is updated a sequence is executed.

Example:

PMCprimo User	Comment
`0.1>  $SPD>PROG1<CR>` `0.1>`	If the value of $SPD is updated, the sequence PROG 1 starts.

Variables can be used in arithmetic expressions involving the standard operators +, -, *, /, %. The % remainder or modulo operator gives the remainder when the left operand is divided by the right. An expression can be used as a command parameter anywhere that a simple variable can be used.

Example: The following example does a move to a position based on the product length ($LEN) and batch size (20) but allowing for bounds wraparound ($BND).

PMCprimo User	Comment
`0.1>  MA(($LEN*20)%$BND)<CR>`	If the product size is 600 counts and the bounds are 5000 counts the command would result in a move to 2000. Note the extra pair of brackets to ensure that multiplication takes place before the modulo operation.

Variables should not be used in place of parameters which include a '+' or '-' sign. In particular, setting variable $A to 2 and typing "**DR$A-**" is not the same as typing "**DR2**"- and will give a syntax error.

In commands which take more than one parameter a variable can be used in place of either or both parameters.

Example (Set scale map):

PMCprimo User	Comment
`1>  $PC=75<CR>`	The variable $PC is set to 75.
`1>  SM$PC,100<CR>`	The scale map is set to 75:100.
`1>`	

Variable and arithmetic expressions must read and calculate before executing. Therefore using variables for commands the execution is slower than using numeric values.

Variables can also be used in logical expressions. These are used with the IF command to allow conditional execution of commands. A logical expression compares two values using the ==, !=, <, >, <=, >= operators to give a result of TRUE or FALSE. It can be combined with other expressions using the && and || operators to give compound expressions. An expression must be enclosed in brackets and it may also be necessary to use further levels of brackets inside the expression to ensure that sub-expressions are evaluated correctly. Ambiguous expressions in sequences may be resolved by PMCprimo; listing a sequence will show the result with brackets inserted as necessary.

The following table shows the valid arithmetic and logical operators:

Precedence	Symbol	Operator	Example	Result
0	( )	Brackets	$a=(5+2)*2	$a=14
1	- + ! ~	Negate Plus Log. negation Bit by bit negation	$a=-2 $a=+2 $b=1;$a=!$b $b=255;$a=~$b	$a=-2 $a=2 $a=0 $a=-256
2	* / %	Multiply Divide Remainder	$a=5*3 $a=5/2 $a=5%2	$a=15 $a=2 $a=1
3	+ -	Plus Minus	$a=5+3 $a=5-3	$a=8 $a=2
4	a<<n a>>n	Logical left shift: Shift 0 from right. The left bits get lost also the sign. Logical right shift: If a has a sign, its shift from left.	$b=8;$a=$b<<2 $b=20;$a=$b>>2	$a=32 $a=5
5	< > <= >= == !=	Less than Greater than Less or equal Greater or equal Equal to Not equal to		
6	& ^ | && ||	Bit by bit AND Bit by bit XOR Bit by bit OR Logical AND Logical OR	$b=250;$a=$b&15 $b=15;$a=$b^2 $b=15;$a=$b|1 $a=(5==4)&&(5>4) $a=(5==4)¦¦(5>4)	$a=10 $a=13 $a=15 $a=0 $a=1

$var=nn Variable assignment

The equals sign ‚=' assigns the value or expression on its right to the variable on its left. Assignment can be used to initialise a variable or to set a variable to a new value based on the value of some other variable. If ‚=' is followed by a command mnemonic which can be queried, the result of the query is assigned to the variable.

Example:

PMCprimo User	Comment
`0.1> $SPD=5000<CR>`	Set variable $SPD to 5000
`0.1> CH1;$POS=DP<CR>`	Set $POS to current position
`0.1>`	

NPName Variable number program assignment

This command displays the program number of the sequence 'Name'. This number is an intern address, which can be used to start a sequence. “**NP**” can be assigned to a variable.

Example:

PMCprimo User	Comment
`0.1> $VAR=NP PRG1<CR>`	The variable $VAR is assigned with the program number 'PRG1'.
`0.1> XS $VAR<CR>`	The sequence PRG1 is started.
`0.1>`	

NTName Variable map number assignment

This command display the map number of the map 'Name'. This number is an intern address, which can be used to start mapping. “**NT**” can assigned to a variable.

Example:

PMCprimo User	Comment
`0.1> $VAR=NT MAP1<CR>`	The variable $VAR is assigned with the map number 'MAP1'.
`0.1> XM $VAR<CR>`	Mapping is executed with MAP1.
`0.1>`	

WV$var Wait for write to variable

This command tells PMCprimo to wait until a value is written to the specified variable before continuing with the command string or sequence. Its not possible to use status- or error variables for this command. Its possible to use the predefined map variable "$MREADY". This variable is used by the intern motion generator.

$var>[PrgName][0|1] Define Trigger variable

The right arrow ‚>' defines the variable on its left as a trigger variable such that each time the variable is updated (the value of the variable must not change) the sequence after ‚>'. It is important to make sure that the triggered commands do not cause the trigger variable to be updated otherwise PMCprimo will enter an endless loop when the trigger variable is first updated.

Enhancement as of **version 1.007:**

With ", 1" after the program name the value must be changed to start the sequence. With ", 0" or nothing after the program name, no change of the value is necessary to start the sequence.

Example:

PMCprimo User		Comment
1>	$SPD>PRG1<CR>	Each time $SPD is updated, the sequence PRG1 is started.

To remove the trigger variable definition the variable and ‚>' should be entered at the end of line with no following commands.

After start up of PMCprimo all trigger variables are inhibited. The command "**EV**" enables the trigger variables.

IA $var[n] Define Variable Array

(Define variable array)

Range n: 1 to 255

Command as of **version 2.000**.

This command defines a variable array. Every element of it can stored extra in the SRAM. Every element is a surrogate variable and can also get global.

Example:

PMCprimo User	Comment
`1.1>  IA $SPD[5]<CR>`	Define variable array $SPD with 5 elements.
`1.1>  $SPD[3]=13`	The third element is assigned an value.
`1.1> $SPD`	Ask for values of variable array
`1.1>`	
`1.1> $SPD[0]=0  (0x00000000)`	
`1.1> $SPD[1]=0  (0x00000000)`	
`1.1> $SPD[2]=0  (0x00000000)`	
`1.1> $SPD[3]=13  (0x0000000D)`	
`1.1> $SPD[4]=0  (0x00000000)`	
`1.1> $SPD[3] > $H10`	Element 3 of $SPD is stored in SRAM.
`1.1>`	

$var>$H1...1000 Store variable in SRAM

In PMCprimo 2+2 and 16+ it is possible to store variables in a battery-backed SRAM. It's possible to store 1000 variables. The bus variables $B1...$B100 can also be stored.

Example: `$VAR > $H12`

The variable '$VAR' is linked with the memory cell '$H12' on the SRAM. The command “**LV**” lists this link.

After starting up it is checked if a lose of data in the SRAM occurred. If this happens the variables are initialised with the values stored by the “**SP**” command.

With redefine a trigger variable storing in the SRAM, the definition is also repealed.

VX List trigger variables

This command displays a list of all the trigger variables. The first character of each line has the following meaning:

E: The trigger variable is enabled
M: The trigger variable is masked (command “**MV**”)
B: The trigger variable is inhibited (command “**BV**”)

Example:

PMCprimo User	Comment
`0.1>  VX<CR>`	List all trigger variables
`M - $SPD>PRG1`	Variable is masked.
`E - $TR>START_PRG`	Variable is enabled
`B - $FRG>PRG2`	Variable is inhibited.
`0.1>`	

MV[$var] Mask trigger variables

This command masks a specified trigger variable, or all trigger variables if no variable name is given. The “**VX**” command shows the current mask/enable state of all trigger variables.

If a value is assigned to a masked trigger variable, after enable with the command “**EV**” the defined sequence is executed.

BV[$var] Inhibit trigger variables

This command inhibits a specified trigger variable, or all trigger variables if no variable name is given. The “**VX**” command shows the current mask/enable state of all trigger variables.

If a value is assigned to a masked trigger variable, after enable with the command “**EV**” the defined sequence is not executed.

EV[$var] Enable trigger variables

This command enables a specified trigger variable, or all trigger variables if no variable name is given. The “**VX**” command shows the current mask/enable state of all trigger variables.

LV List variable value

If "**LV**" is entered, PMCprimo lists all the currently defined with their values. If the name of the variable itself is entered, PMCprimo lists the value of the specified variable.

Example 1:

PMCprimo User	Comment
`0.1> $SPD<CR>`	List value of $SPD
`$SPD=5000 (0x00001388)`	Current value of $SPD is 5000
`0.1>`	

Example 2:

PMCprimo User	Comment
`0.1> LV<CR>`	List all variable values
`$BCD=12`	
`$SPD=5000`	
`$TR=75`	
`0.1>`	

Example 3:

PMCprimo User	Comment
`0.1> IA $VAR[5]<CR>`	Define variable array with 5 elements
`0.1> LV<CR>`	List all variables
`$VAR [200]`	
`0.1> $VAR`	List variables of array
`0.1> $VAR[0]=0 (0x00000000)`	
`0.1> $VAR[1]=0 (0x00000000)`	
`0.1> $VAR[2]=0 (0x00000000)`	
`0.1> $VAR[3]=0 (0x00000000)`	
`0.1> $VAR[4]=0 (0x00000000)`	

LB List bus variable value

This command lists the predefined bus variables. It exists 100 bus variables with the following syntax:

$Bn n: from 1 to 100 (1 to 108 as of **version 1.008**) range: ±32.768 (2 Byte signed)

This bus variables are the interface for extern devices for example an operators panel.

They can also be defined as trigger variables or can be used as normal variables.

Example:

PMCprimo User	Comment
`0.1> $B12<CR>`	List value of $B12
`$B12=8 (0x00000008)`	The value of $B12 is 8.
`0.1>`	

LP List map variable value

This command lists the predefined variables for the internal motion generator (see therefore "PMCprimo Motion Generator **PMotion** user manual").

Enhancement as of **version 2.002**:
The variables $MMSB and $MSSB are also listed with the command.

$Ft.n Error variable

Range for t: 0 to 59, (depends on number of nodes) or no value
Range for n: 1 to 3 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 20 adjustable with command “**CD**” (PMCprimo 16+)
1 to 10 adjustable with command “**CD**” (PMCprimo Drive2)

For every channel exists a predefined error variable. The name is composed with the following details:

$FNode.Channel

The error variable stores the error number of the last error. An error variable can be defined as a trigger variable. It can made a value assignment to an error variable. See chapter 9.2 on page 192 for the different values.

Enhancement as of **version 1.007:**
It is possible to make an assignment to the error variable e.g. $F0.1=0. This includes a reset for the 7-segment display.

Enhancement as of **version 2.000**:
In the error variable $F0.0 the last error number of Host, CAN, mapgenerator and compiler is stored.
The error message “Program is not defined” and “Cannot enter or execute program while it is in use” makes no error number for $F0.0, because if $F0.0 is defined as a trigger variable a never ending loop can happen.

$St.n Motor status variable

Range t: 0 to 59, (depends on number of nodes) or no value
Range n: 1 to 3 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 20 adjustable with command CD (PMCprimo 16+)

For every channel exists a predefined motor status variable. The name is composed with following details:

$SNode.Channel

The motor status variable stores the actual status of the channel. A motor status variable can be defined as a trigger variable. It can made a value assignment to a motor status variable. See chapter 9.1on page 191 for the different values.

$Wt.n Set wait status variable

Range for t: 0 to 59, (depends on number of nodes) or no value
Range for n: 1 to 3 (PMCprimo Drive)
1 to 4 (PMCprimo 2+2)
1 to 20 adjustable with command “**CD**” (PMCprimo 16+)
1 to 10 adjustable with command “**CD**” (PMCprimo Drive2)

For every channel exists a predefined wait status variable. The name is composed with following details:

$WNode**.**Channel

The wait status variable stores the actual status of the channel. A wait status variable can defined as a trigger variable. It can made a value assignment to a wait status variable. See chapter 9.1on page 191 for the different values.

8.16 Commands for CANopen network

Messages are send in a CANopen network with follow objects:

SDO Service Data Object. It's a data element with low priority. It is used to configure a node. The transfer is with reconfirming.

PDO Process Data Object. It's a data element with high priority. It's for real time data exchange. The transfer is without reconfirming.

It is possible to write and read SDO's with special commands:

QSt,i,s,c,n Send SDO

Parameter t: CAN-node
Parameter i: Index
Parameter s: Sub index
Parameter c: Number of Bytes
Parameter n: value

With this command, PMCprimo can access an object directory of a CAN device via SDO, for example to write demand values or to configure the device. Please read the documentation of the device for setting the configuration parameters. The read value can be allocated to a variable.

QRt,i,s,c Read SDO

Parameter t: CAN-node
Parameter i: Index
Parameter s: Sub index
Parameter c: Number of Bytes

With this command, PMCprimo can access an object directory of a CAN device via SDO, for example to read actual values or to configure the device. Please read the documentation of the device for setting the configuration parameters. The read value can be allocated to a variable.

PDt Position from drive

Range t: 0 to 59 (max. 60 number of nodes)

With this command it is possible to control a PMCtendo DD4 with CANbus on channel 2 and 3 (PMCprimo Drive) or channel 3 and 4 (PMCprimo 2+2) in VM0 modus. The position is returned from CANbus or from the encoder (see command FS page 151). PD0 redefines the definition.

Enhancement as of **version 1.008**:

The command "**PD**" can be used in PMCprimo 2+2 on all 4 channels. Now it is possible to control 4 PMCtendo DD4 with CAN-Bus in 4ms cycle time.

Enhancement as of **version 2.001**:

With PMCprimo 16+ the command "**PD**" can used on max. 16 channels at the same time. The cycle time depends on the number of PMCtendo DD4 in the CANbus.

With value 0 the definition will be canceled and the connection via CAN disconnected.

	Useable addresses	**Number per CAN**
PMCprimo Drive	2+3	2 only CAN
PMCprimo Drive2	2-10	(4[*1]) 8[*2] per CAN-network
PMCprimo 2+2	1-4	4
PMCprimo 16+	1-16	(4[*1]) 8[*2] per CAN-network

([*1] till Version 2.004) ([*2] from Version 2.005)

Cycle time:

In the main network, mode standalone, the cycle time is abjustable with the "**CD**" command. In mode "Host+Node" the cycle time is always 4 ms. In extended CAN-network, the cycle time will be choosen automatically according to the following table:

PD	**500 KBit**	**1 MBit**
1-2	1 ms	1 ms
3-4	2 ms	1 ms
5-8	4 ms	2 ms

9 Status- and error messages

9.1 Status messages

XX This section gives the PMCprimo messages in various circumstances:

> Normal prompt.
This is the prompt character in position control mode. PMCprimo is ready for the next move command.

: Motor off prompt.
This is the prompt for position control loop is open.

A Executing map alignment.
This channel is starting to enter position mapping using an alignment move (MW bit 0 set to 0). If this prompt is visible the "**XM**" command is still executing.

C Executing software clutch.
This channel is starting to enter position mapping using the software clutch (MW bit 0 is set to 1). If this prompt is visible the "**XM**" command is still executing.

I Initialising.
The channel is executing the "**IN**" initialise command.

M Moving.
The channel is executing a normal trapezoidal or sinusoidal move.

S Stopping.
The channel is executing a normal controlled stop.

V Velocity control mode.
The channel is executing a constant velocity move.

W Waiting.
The channel is waiting for some condition before continuing.

X Executing a position mapping.
This channel is linked to another master axis and is executing a position mapping (and has completed the clutch or alignment phase). The ‚X' prompt is used to indicate that the channel is cross-linked to another channel, since the ‚M' prompt is used for normal moves.

nnnE Enter sequence (see "**ES**" command on page 54).

nnnM Enter map definition (see "**EM**" command on page 72)

9.2 Error messages

This section describes the various messages displayed by PMCprimo. The language parameter LG is set 1.

Code	Error
0	Internal Error
1	To many threads in use
2	Command not available
3	Cannot execute <MA> while motor off
4	Position limit exceeded
5	High position limit exceeded
6	Low position limit exceeded
7	Cannot execute <MA> while motor is running
8	Cannot execute <MR> while motor off
9	Cannot execute <MR> while motor is running
10	Cannot execute <VC> while motor off
11	Cannot execute <VC> while motor is running
12	Attempt to divide by zero
13	The parallel execution must be end (command CE) to execute a program
14	Stack overflow
15	Mapgenerator already in use
16	Query inputs
17	Query outputs
18	Error mapgenerator: Variable <$MNAME> not defined
19	Error mapgenerator: Variable <$MNPT> not defined or zero
20	Variable <…> allready in use by <WV>
21	input no counter input
22	input no gate input
23	Memory full
24	Motor position error
25	Reset input not possible
26	No trigger variable defined
27	Cannot enter or execute program <…> while it is in use
28	Target position outside limits
29	Motor timeout. No encoder signals
30	Encoder counter read error. Wrong position deleted and interpolated position used
31	Cannot execute <XM> while motor off

Code	Error
32	… bytes ram memory free
33	… bytes flash memory free
34	Interrupt took to much time
35	Cannot execute program <…> because it is not defined
36	Execute <ML> before
37	Reference error outside limits
38	Reference correction overrun
39	Reference timeout
40	Reference error reduced to maximum value
41	Program <…> not activ
42	Speed mapping only linear possible
43	Active processes:
45	Active program: <…> node: … line: …
46	Masterbound:
47	Input number greater than 8
48	Input group not defined
49	Output number greater than 8
50	Output group not defined
51	Number of values:
52	Output is already defined as 'error output'
53	Output is already defined as 'position trigger output'
54	Output is already defined as 'bound overflow output'
55	Undefined error
56	Limit switch detected
57	Input is already defined
58	Input is not defined as 'limit switch'
59	At least 2 pairs of values for mapgeneration must be defined
60	Error mapgenerator: Variable $MS… not defined
61	Error mapgenerator: Wrong value for $MX…
62	Error mapgenerator: Wrong value for $MY…
64	Variable for mapgeneration not defined
65	Fatal error compiler
66	Parameter <…> out of range
67	Cannot execute ML while channel in mapping
68	Error mapgenerator: Wrong value for $MM…!
69	Function input not possible
70	Gate input not possible

Code	Error
71	Counter input not possible
72	MLn: Cannot link channel to itself
73	Cannot access channel <…> (command <…>)
74	Timout for position update
75	Cannot execute <QR> (CAN-Bus not avaiable?)
76	Cannot execute <PC> because AO is defined
77	Output is already defined as AO
78	'Position snap shot' already defined at this channel
79	Input is no 'fast input'
80	The 'fast input' is already defined
81	Reference input not defined
82	Command IB not possible, because channel virtuell und motor off
83	Cannot execute LW while channel is defined by ML
84	If output defined, don't use <SO> or <CO>
85	Ouput is already defined as 'reference output'
86	Ouput is already defined as 'analogue limit error output'
87	Output not defined
88	Ouput is already defined as 'reference correction output'
89	Zero marker input is defined
90	'Position snap shot' already defined at this channel
91	'Reference input' already defined at this channel
92	Input is already used from another channel
93	Fast input not defined as 'motor positon snapshot'
94	Fast input not defined as 'reference input'
95	The output is still defined with RA
96	Error IP: Limit of 2*SV exceeded
97	Error IP: SS > SV
98	Error IP: Direction of velocity must'n change
99	Cannot execute <XR> while motor off
100	Cannot execute <XR> while motor is running
101	Error mapgenerator: Variable $MF… not defined
102	Error mapgenerator: Variable $MW… not defined
103	Error mapgenerator: Variable $MA… not defined
104	Error mapgenerator: Variable $MZ… not defined
105	Error mapgenerator: Variable $MX… not defined
106	Error mapgenerator: Variable $MY… not defined
107	Error mapgenerator: Variable $MX… > $MY…

Code	Error
108	Error mapgenerator: Variable $MB… not defined
109	Error mapgenerator: Variable $MC… not defined
110	Error mapgenerator: Segment Nr….: the summ must be 100%
111	Mapgenerator not enabled !
112	Mapgenerator started !
113	Mapgenerator ready
114	Output is already defined as <JF>!
115	Command <XR> or <XM> was breaking off
116	Operator panel varibale not defined
117	Analogue input low limit exceeded
118	Analogue input high limit exceeded
119	Execute <ML> before <XV>
120	Input is already defined as 'position snap shot'
121	Input is already defined as 'reference input'
122	<PC> not possible while initialisation
123	<AO> not possible while <PC>
124	<VM1> not possible while AO defined
125	<IP> with <IS7> only possible if channel in mapping
126	Program <…> not defined
127	Wrong output number
128	Wrong output group
129	Value of shiftregister must be between 1 and 32
130	<FC> only possible if <MO>
131	<VM> only possible if <MO>
132	No clutching without softwareclutch and moving master
133	Cannot execute <VC> while channel in mapping
134	Cannot execute <WA> while it is in use
135	Cannot execute <WR> while it is in use
136	Cannot execute <WF> while it is in use
137	Cannot execute <WB> while it is in use
138	Cannot execute <WC> while it is in use
139	Channel is only a virtual channel
140	Map not defined
141	Mastervalue not in map
142	Channel not defined
143	Variable not defined
144	Wrong nodenumber in input term

Code	Error
145	DI not defined
146	Fast input already defined for channel
149	Maximum 8 mastervalues with the CAN-Bus allowed
150	Cannot execute <UL> while mapping
151	Execute <ML> before <XM>
152	Externer Master for <XR> not allowed
153	Checksum error, data lost
154	Trace mode already in use
155	Possible baudrates: 9600 14400 19200 38400 57600 115200
156	There is no value trace mode
157	While trace mode <TW> not allowed
158	Channel >4 doesn't exists for TW
159	Wrong datatype for TW-command
160	No program defined
161	Reset is done
162	Modus <…> can't used for <TC>
163	Wrong value <…> for command <TC>
164	Softwareupdate successful
165	Attention!!! error softwareupdate
166	RESTORING DATA...
167	READY RESTORING DATA
168	STORING DATA
169	READY STORING DATA
171	Use <NL> for softwaredifferential
172	Drive error: heat sink temperatur to high
173	Drive error: overvoltage
174	Drive error: feedback (cable break, short circuit, short to ground)
175	Drive error: undervoltage
176	Drive error: motor temperatur
177	Drive error: internal aux. voltage not OK
178	Drive error: overspeed
179	Drive error: EEprom checksum error
180	Drive error: Flash-Eprom checksum error
181	Drive error: brake (cable break, short circuit, short to ground)
182	Drive error: motor phase missing
183	Drive error: internal temperature to high
184	Drive error: output stage not OK

Code	Error
185	Drive error: I2t max value exceeded
186	Drive error: supply power - BTB/RTO
187	Drive error: A/D-Converter defective
188	Drive error: regen circuit faulty
189	Drive error: supply phase missing
190	Drive error: system error (system software)
191	Drive warning: I2t-threshold exceeded
192	Drive warning: preset regen power reached
193	Drive warning: watchdog for expansion card activated
194	Drive warning: supply phase missing
195	Drive warning: Hiperface: motor default values were loaded
196	Drive warning: expansion card not OK
197	Drive warning: enable signal low
198	Error CAN-Bus: safety line not switched
199	Error CAN-Bus: writing PDO: CAN not in mode operational
200	Error CAN-Bus: writing in queue
201	Error CAN-Bus: the safety line was switched from one node
202	Error CAN-Bus: 12V-power supply switched off
203	Error CAN-Bus: transmitbuffer overflow : messages are lost
204	Error CAN-Bus: receivebuffer overflow: messages are lost
205	Error CAN-Bus: too many CAN errors: Can-controller busoff
206	Error CAN-Bus: CAN-Receivebuffer overflow
207	Error CAN-Bus: transmit error
208	Error CAN-Bus: watchdog expired
209	Error CAN-Bus: error while transmitting SDO: Timeout
210	Error CAN-Bus: error while transmitting SDO
211	Error CAN-Bus: unknown message
212	Error CAN-Bus: too many CAN errors: shuting down CAN bus
213	CAN-Bus deactivated !
214	CAN node …: firmware version does not match to host
215	Error CAN: cannot activate servo controller on CAN bus
216	Error CAN: cannot deactivate servo controller on CAN bus
217	PD command on channel … not possible !
218	Error CAN-Bus: Node guarding error, node … !
219	Attention: lost of data in battery buffered memory
220	Battery buffered memory not available
221	Map wasn't transfer to channel (command TM)

Code	Error
223	Mapping not activ !
224	Error from servo controller on CAN bus
225	Attention: Bit 1 of FW was set to 1, because RJ>SB
226	Wrong value from drive command for variable assignment
227	No data in flash stored !
228	To less flash-memory !
229	Error while storing data !
230	Buffer for storing reference position is full !
231	Direction has changed ! The buffer with reference position must delete
232	Possible baudrates: 96000 19200(XOn/XOff) 38400(XOn/XOff)
233	Output is already defined as 'velocity trigger output'
234	To many PLC-tasks active
235	No correct datas for TR
236	No output definition for CAN-IO modul
237	TD…,… is not defined!
238	The output is already defined as 'outside window output'
239	Cannot execute ID, while axis is not in status 'PC'
240	Parameter <…> for command <…> out of range, if bit 4 of RW is set
241	Parameter <…> for command <…> out of range, if bit 4 of RW is not set
242	Parameter <…> for command <…> out of range, if bit 6 of RW is set
243	Parameter <…> for command <…> out of range, if bit 6 of RW is not set
244	Value of RC<1000 not allowed, because bit 7 of RW is set to 0
245	Value of RV>200 not allowed, because bit 5 von RW is set to 0.
246	Bit 4 of RW is set to 1, because bit 6 was set to 1.
247	Intern error PLC
248	Cannot execute drive command, because node is no drive
249	Drive error: Commutation error
250	Variable $…[…] is not defined
251	Node number not defined (command <…)
252	Node number … is already in use (command <…)
253	No CAN-device found
254	Cannot execute drive command, because buffer overflow
255	Warning: SRam battery low
256	Drive error: Enable switched on before AS-option
257	Drive error: Earth short ciruit
258	Drive warning from CAN
259	.. bytes CompactFlash memory free

Code	Error
260	Cannot execute <XM> while motor is running
261	Command QA0:CAN-Adr can only done on the host
262	The analog output <…> is not available! (command <…>)
273	PD command not possible! PMCtendo DD4 version wrong(< 4.94)
274	Node …, Drive command not completed !
276	Battery buffered memory is used by SoftPLC
277	Die maximum possible position (SB*SM) is greater than the position limit of … (command <…>)
278	The predefined map 'LINEAR' can't be changed

9.3 Status Codes

This section gives the numeric values for the status codes which are placed in the status variables.

0 Idle mode (PC)

8 Motor off (MO)

66 Waiting for clutching in at the beginning of mapping (XM)

67 Executing software clutch (XM and MW bit 0 set to 1). (C).

68 Executing alignment move (XM and MW bit 0 set to 0) (A).

256 Constant velocity mode (VC)

512 Moving (MA,MR)

2048 Executing a position mapping (= Map) active (XM)

4096 Stopping (ST)

8192 Initialising (IN,IB)

Every bit of a wait command belongs to a bit of the wait status variable $W.

WI: Bit 2

WA: Bit 3

WR: Bit 4

WF: Bit 5

WB: Bit 6

WC: Bit 7

10 Interface PMCprimo <-> machine

10.1 Notes on Installation

Care should be taken in all installations to protect PMCprimo from high voltages and to minimize electrical noise on signal and power supply lines. Pilz can accept no responsibility for problems arising from poor installation. Please refer to the PMCprimo Installation manual for more information.

A digital servo controller relies on the position information from its incremental encoder, and any noise on the encoder signals can give rise to errors in the absolute position. Care must be taken in installation of PMCprimo and the encoders to minimize any noise on the encoder signal lines. PMCprimo has full optical isolation on all the encoder signals, and require encoders with complementary line driver outputs. The encoder input interface has a differential input stage for use with such encoders, providing high rejection of common-mode noise. In addition, spurious signals on one encoder track produce both an up and a down count, and thus cancel out. However, in particularly electrically noisy environments it is still possible to get position counting errors. Noise is reduced by using encoders with line driver outputs. Where the environment is electrically noisy, or where the system will be used continuously and reliability is important, it is possible to set up the system such that its position is continuously operating, unless the system was stopped at regular intervals to reinitialise the absolute position.

The digital input and output lines are also fully isolated from the machine or plant, both for protection and to allow 24V signals to be used. This provides greater noise immunity and allows direct interfacing to industrial control equipment such as a programmable logic controller (PLC). Isolation is also available as an option on the analogue output signals if required.

All electrical connections, such like suspense supply, the connection for the drive and the in- and outputs are done with screw terminal blocks.

10.2 Safety

The PMCprimo system provides many safety facilities, and it is recommended that these are used in addition to external safety systems such as hardwired limit switches.

PMCprimo can accept no responsibility for problems due to incorrect use of the safety features provided.

The safety features of the system are provided for very good reasons! It is important to understand the operation of all these facilities, as it is possible to do fast amounts of damage to both machinery and people with high performance motors and drives. It is not sufficient to decide that these facilities are not relevant to a particular application; they are provided to monitor the correct operation of the whole system, and if the system gives an error then it is telling you something important. The relevant commands are listed here.

SE Set maximum position error

TO Set timeout

LH Set high position limit

LL Set low position limit

DL Define limit switch inputs

OB Define motor brake output

BD Set brake delay time

Please read thoroughly the descriptions of these commands at least, if no others.

10.3 Position Encoder

The system is designed for use with digital incremental position encoders. These encoders provide two signals in quadrature (one is phase shifted by 90° relative to the other). The system can monitor these signals and determine both the direction and distance of any movement. The direction is defined by which signal leads the other. The normal definition is such that the track A encoder input leads the track B input for movement in the positive direction.

The system generates four counts for each complete cycle of the input signals, such that an encoder with 1000 lines per turn is seen as generating 4000 counts per turn. The encoder input signals are all fully isolated. The standard systems are designed for use with encoders having complementary line driver outputs, for maximum noise immunity. The position encoder feedback is fundamental to the correct operation of the system, and so all precautions against noise are justified. It is also possible to use SSI encoders. Hiperface is only possible with ' PMCprimo Drive'. Moreover it is possible to use a CANopen encoder. In a network the bus number of the encoder must be the same as the node number of the axis. If PMCprimo is configured to 'standalone' the bus number of the encoder is arbitrary, because PMCprimo detect it automatic.

10.4 Demand Output (PMCprimo 16+)

The normal demand output signal to the high power motor drive is an analogue signal with a range of ±10V, at 12 bits resolution. This output is switched directly to 0V in the motor off state by a reed relay for each axis. The PMCprimo is normally connected such that a positive demand output signal causes the motor to move in the positive direction.

In the PMCprimo Drive the interface for the demand output is digital.

10.5 Auxiliary Output

Every analogue output of PMCprimo which is used normally as demand output can be used as auxiliary output with a range of ±10V. The definition is made with the "**AO**" command (see page 103). The signal is appointed with the commands "**SF**", "**KM**" and "**OM**".

10.6 Serial Communications

The serial link uses RS-232 signal levels as standard. The serial word format used is 8 data bits, 1 stop bit, and no parity. The baud rate is by the PMCprimo Drive fixed at 9600 baud. By the PMCprimo 2+2 and 16+ the baud rate can be set with the command "**BD**" to a maximal value of 19200 baud. The baud rate can't be saved and must set therefore before communication for example in "**AS**" (see page 61).

The serial interface is buffered in software and echoes back the characters as they are received. All the PMCprimo systems use xon/xoff software handshake by default on the programming terminal port, but they also support hardware handshake using RTS and CTS signals.

10.7 CANopen Interface

The communication with PMCprimo and the PC can also made with a CANopen interface.
Therefore a CAN bus card in the PC is needed.

Besides it can access to the bus variables $B1 to $B100 by CAN bus. The follow SDO's must set for it:

CMS	Index	Sub index	Comment	Default	Range	Data length
SDO	0x3000	0	Variable $b1	0	-32768..32767	Integer16
		1	Variable $b2	0	-32768..32767	Integer16
		...	...			
		99	Variable $b100	0	-32768..32767	Integer16

Enhancement as of **version 2.005**:
It is now possible to use the Sontheim CAN-I/O Modul DIO32 (16 inputs and 16 outputs).

10.8 Analogue Inputs

The signal inputs are multiplexed, and the multiplexer output is buffered with a differential amplifier. The analogue inputs have a range of ±10V. The analogue signal levels are converted to digital values at 12 bits resolution. The "**DA**" command displays the analogue input signal value for the current channel. The analogue input can used for analogue control (see chapter 8.14).

10.9 Digital In- and Outputs

PMCprimo has a fixed number of in- and outputs depending on the hardware. The in- and outputs are divided in groups (bytes) with 8 lines. The digital input and output lines are also fully isolated from the machine or plant, both for protection and to allow 24V signals to be used.

PMCprimo Drive/2	3 input groups, at what only 4 lines of the second group can be used. The third input group is a virtual and connected by software with the virtual outputs of group two. Only the inputs 1 and 2 are fast inputs and can be used for the “**DR**” and “**PS**” commands. 2 output groups, at what the second group is virtual. As of **version 2.004**: Now 16 virtual inputs and 16 outputs are available. The output group 2 is linked to input group 3 to be compatible with existing programs. The output group 3 is linked also to input group 3. The additional outputs are then linked on group 4 together. The virtual inputs and outputs have the same numbering with all systems
PMCprimo 2+2:	3 input groups, at what the third input group is a virtual and connected by software with the virtual outputs of group two. Only the inputs 1 to 4 of input group 1 are fast inputs and can be used for the “**DR**” and “**PS**” commands. 3 output groups, at what the second group is virtual.
PMCprimo 16+:	3 input groups, at what the third input group is a virtual and connected by software with the virtual outputs of group two. Only the inputs 1 to 4 of input group 1 are fast inputs and can be used for the “**DR**” and “**PS**” commands. 3 output groups, at what the second group is virtual.

It is possible to use a CAN-IO module in the CANopen bus of PMCprimo. The inputs have always the node number 61 and can be used as normal inputs.

PMCprimo User	Comment
`0.1> SO61.1:1`	CAN-IO 1.1 is set
`0.1> DI61.1:2,PRG`	For CAN-IO 1.2 the sequence 'PRG' is defined

CAN-IO's can be used in all operation modes of PMCprimo:

1. Standalone (only CAN-IO`s and CAN-encoder useable)
2. Host+axis (networked)

The follow commands can be used for CAN-IO's:

- MG Define input mask group (see page 139)
- BG Define input inhibit group (see page 140)
- DI Define function input (see page 138)
- II If input true do command line (see page 125)
- WI Wait for input line (see page 88)
- UI Undefine input line (see page 129)
- MI Mask input line (see page 127)
- EI Enable input line (see page 129)
- LI List input line definitions (see page 150)
- RI Read input line (see page 124)
- BI Inhibit function input (see page 128)
- SO Set output line (see page 123)
- LO List output line definitions (see page 150)
- RO Read output line (see page 125)
- CO Reset output line (see page 124)
- IO If output true do command line (see page 126)

10.10 Drive commands

With PMCprimo Drive it is possible to set commands for the drive direct form the PMCprimo shell. The drive commands must be put in quotation marks.

Example:

PMCprimo User	Comment
`0.1> "DIR"`	Drive command DIR is set from PMCprimo shell.
`0.1> DIR`	output from Drive

It is possible to assign a drive command to a variable. If the drive returns no value it comes to a error message.

Example:

PMCprimo User	Comment
`0.1: $VAR="DIR"<CR>`	Drive command DIR is set to the variable $VAR.
`0.1. $VAR<CR>`	Display $VAR
`0.1: $VAR=1 (0x00000001)`	

Saving with the command "**SP**" the drive parameters are stored in the sequence with the following name:

DRIVE_t t: node number

Enhancement as of **version 2.000**:
Assignment of a variable to a drive command:
e.g.: „GV" = $VAR. The value of $VAR is divided intern with 1000 in order to get a floating point number. For commands who needs an integer the value must multiplied always with 1000.

Enhancement as of **version 2.007**:
Until now the drive commands are done in this way:
"CMD"=$VAR
The system alos writes a floating point value. If $VAR=2100 then the command "CMD" was written.
$VAR="CMD"
If the command returns a floating point value (of dot was returned) then the value was multiplied with 1000.
GV 2.1 gives $VAR=2100 but GV 2 gives $VAR=2000
With the command "CMD"= VAR the value is only divided with 1000 if the command needs a floating point value.
$VAR=2100
"GV"=$VAR gives "GV 2.1"
"PEMAX"=$VAR gives "PE 2100"

QA n:m **ASCII channel for PMCtendo DD4**

n=0: PMCtendo DD4 in CAN-Net, command must execute on the host node.
n=1: PMCtendo DD4 in CANOpen
Range m: 1 to 61

With the command QA you can open an ASCII channel to communicate with a PMCtendo DD4 drive. With the parameter n the CAN-Net (page 210) is selected. The parameter m specifies the address of the PMCtendo DD4. It can communicate on the actual channel with the drive commands described before. The command is channel specific so you can communicate on every channel command with a different PMCtendo DD4.

10.11 Operate modes

Every device in the network is with its node number identified. Every number can be used only one time.

Device	Host	Node	CAN-I/O-Module
Node-No.	0 (firm)	1 – 60	61 (firm)

10.11.1 Host

The host administers and saves all sequences, maps and variables in the network. All devices are programmed by the interface of the host.

It can connect an operator panel with the host.

This configuration joins both possibilities in one device.

Only one host is allowed in the network!

The user can choose which device the PMCprimo Host + Node is. It depends not on a physical position in the network. It's advisable to take the node with the least utilization.

10.11.2 Node

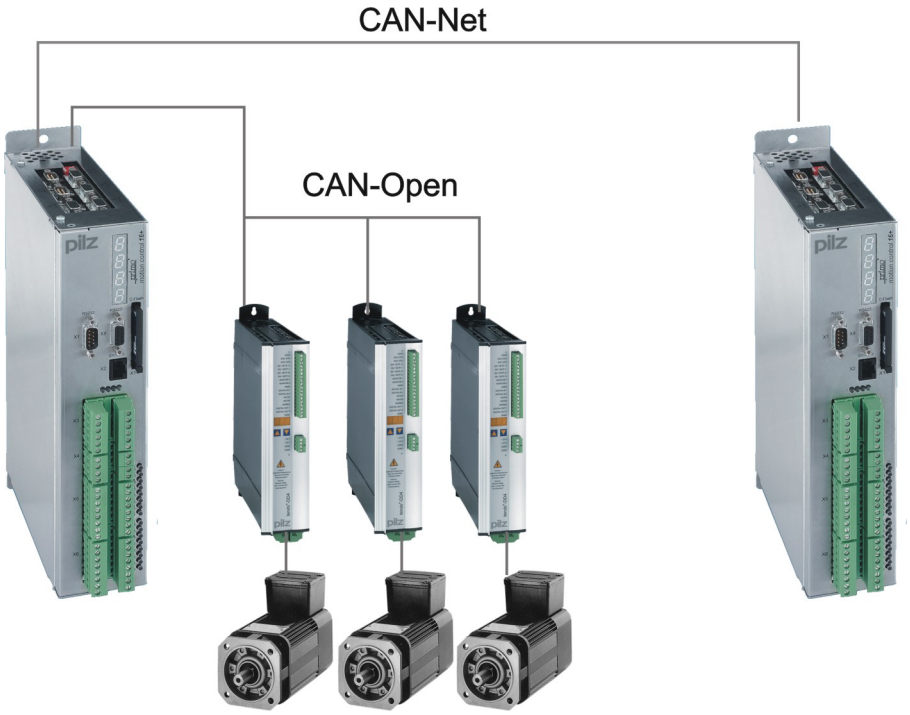
The nodes get their sequences from the host. Every node has its own I/O and axis.

10.11.3 CAN-Net and CAN-Open

As of **version 2.000** PMCprimo supports more than one CAN-Net.

The main net, which connect the PMCprimo devices is described as **CAN-Net**.

The sub net, which connects a PMCprimo device with some drive devices like PMCtendo DD4 is described as **CAN-Open**.



$

$var ... 182
$var> ... 183
$var>$H1...1000 ... 184

?

?? ... 168

A

AA ... 80
AB ... 33
Abort ... 33
Abort command execution ... 60
AC ... 175
Acceleration ... 39
acceleration feed forward ... 98
Actual position ... 161
Actual reference error ... 161
AD ... 175
AE.. ... 149
AH ... 178
AI ... 175
AL ... 178
AM ... 174
Analogue Control ... 174
Analogue control mode ... 174
Analogue control options word ... 176
Analogue control proportional gain ... 175
Analogue control setpoint ... 175
Analogue differential control gain ... 175
Analogue input ... 162
analogue input high limit ... 178
Analogue input low limit ... 178
Analogue Inputs ... 204
Analogue integral control gain ... 175
Analogue limit error output ... 149
Analogue range distance ... 177
AO ... 103
AP ... 175
AR ... 177
AS ... 61
ASCII channel ... 208
assignment Variable ... 182
AT ... 177
Auxiliary Output ... 203
AV ... 79
AW ... 176
AX ... 60

B

BA.. ... 172
Backlash compensation distance ... 45
Base advance time constant ... 173
Baud rate modbus ... 21
BC ... 109
BD ... 21
BG ... 140
BI ... 128
Bit by bit AND ... 181
Bit by bit negation ... 181
Bit by bit OR ... 181
Bit by bit XOR ... 181
BK ... 61
BL.. ... 45
BM ... 21
BN ... 142
BO ... 147
Bound overflow output ... 147
BR ... 79
Brackets ... 181
Break out of sequence ... 61
BT ... 173
BV ... 185

C

Calculate initial ratio from analogue range distances .. 178
CAN-Net ... 210
CAN-Open ... 210
CANopen Interface ... 204
CD ... 16
CE ... 62
CH ... 61
Change channel ... 61
Change the operating mode ... 16
Change to parallel mode ... 62
CI ... 86
CL ... 86
Clear output line n in byte g ... 124
Clutch window ... 86
CO... ... 124
Command Execution ... 14
Command line ... 14
Configuration Commands ... 135
Configuration word for trace mode ... 165
Continous display mode ... 163
continous reference mode on/off ... 109
Control word ... 48
CP ... 62
CR ... 178
Creep distance ... 41
CT ... 85
CW ... 48

D

DA ... 162
DB ... 147

DC 40
DD 161
DE 143
Deceleration 40
Deceleration for AB command 40
Deferred reference adjustment position 115
Define analogue input type 177
Define analogue limit error output 149
Define analogue range distance 177
Define bound overflow output 147
Define error output 143
Define function input 138
Define hysterese for position trigger output 146
Define hysterese for velocity trigger output 147
Define inhibit input line 142
Define input inhibit group 140
Define input mask group 139
Define masked input line 141
Define master axis analogue range distance 177
Define output for refrence adjustment 119
Define outside window output 147
Define position snapshot input 137
Define position trigger output 144
Define reference accepted output 119
Define reference input 121
Define reference reject output 148
Define timer/counter clock input 133
Define timer/counter gate input 134
Define timer/counter output 130
Define timer/counter reset input 134
Define trigger variable 183
Define Variable Array 184
Define velocity trigger output 146
Define zero marker input on/off 115
Demand Output 203
Demand position 161
DF 161
DG 161
DI 138
Digital In- and Outputs 205
Display actual position 161
Display actual reference error 161
Display analogue input 162
Display demand position 161
Display following error 161
Display last error 169
Display last value of trace mode 168
Display length reference signal false 118
Display length reference signal true 118
Display mapped axis bound position 162
Display mode off 162
Display reference error 161
Display snapshot position data 162
Display time 162
Display trigger variables 185
Display velocity 161
Divide 181
DL 136
DM 163
DO 162
DP 161
DR 121
Drive commands 207
DS... 162
DT 162
DV 161
DZ 115

E

EI 129
EL 59
Else - execute commands if expression is false 59
EM 72
Emergency stop 33
EN 129
Enable function input 129
Enable function input with priority 129
Enable trace mode 164
Enable trigger variables 185
Encoder filter 47
End of enter map 72
End of parallel mode 62
End of sequence 55
End repeat 58
End wait state 92
Enter map 72
Enter position control mode 27
Enter sequence 54
Equal to 181
ER 58
Error messages 192
error options word 96
Error variable 187
ES 54
EV 185
EW 96
Execute analogue range distance initialisation 178
Execute commands if expression is true 59
Execute commands if program is activ 60
Execute map 74
Execute map virtual 75
Execute sequence 57
Execute sequence parallel 57

F

FC 159
FE 161
Feedback change encoder 159
FH 117
filter reference error 112
FL 117
FM 24
Following error 161
FR 112

FS... 151
Function input 138
FW... 116

G

GA 63
Gain commands 98
GE 63
Get wraparound offset value 162
GF 63
Global abort 63
Global abort execution 63
Global motor off 63
Global stop 63
Global wait off 63
GM 162
Greater than 181
GS 63
GW 162
GX 63

H

Help display 168
Hiperface 151, 159
Host + Node 209
HW 20
hysterese for position trigger output 146
hysterese for velocity trigger output 147

I

IA 184
IB... 36
ID 37
IF 59
If input true do command line 125
if motor status true do command line 126
If output true do command line 126
II... 125
IN... 35
Increment select code 46
Increment selected parameter 47
Inhibbit trigger variables 185
Inhibit function input 128
Initialise position 35
Initialize demand signal offset 37
Initialize position and bounds 36
Input debounce time 147
Introduction 7
IO... 126
IP 47
IS 46
IT 126
IX 60

J

JB 120
JF 120

K

KA 100
KD 99
KF 100
KI.. 99
KM 101
KP 99
KV 99

L

LA 24
Language 169
Last error 169
Last value of trace mode 168
LB 186
LC 132
LE 169
length of alignment move 75
Less than 181
LG... 169
LH.. 94
LI... 150
List all parameters 24
List busvariable value 186
List counter value 132
List input line definitions 150
List map 73
List mapvariable value 186
List output line definitions 150
List sequence 56
List variable value 186
LL 95
LM 73
LO 150
Log. negation 181
Logical AND 181
Logical left shift 181
Logical OR 181
Logical right shift 181
LOOP 55
Loop for sequence 55
LP 186
LR 112
LS... 56
LV 186
LW 83

M

MA 28

Map base advance ... 172
map link options word ... 83
Map link slave axis to master axis ... 76
Map link to differential axis ... 76
map options word ... 81
Mapped axis bound position ... 162
Mask function input ... 127
Mask trigger variables ... 185
Master axis analogue range distance ... 177
Master scale factor ... 160
Masterposition timeout ... 86
Maximum reference correction ... 111
MB ... 77
ME ... 63
MF ... 77
MG ... 139
MI... ... 127
Minus ... 181
ML ... 76
MM ... 177
MN ... 141
MO ... 27
Mode commands ... 27
Mode off ... 162
Motor off ... 27
motor status ... 91
Motor status variable ... 187
motorstatus ... 126
Move at constant velocity ... 34
Move commands ... 28
Move relative ... 31
Move to absolute position ... 28
MR ... 31
MS ... 160
MT ... 86
MU ... 21
Multiply ... 181
MV ... 185
MW ... 81

N

NB ... 159
Negate ... 181
NL ... 76
NM ... 72
Node ... 209
Not Equal to ... 181
Notes on Installation ... 201
NP ... 182
NS ... 55
NT ... 182
number of bits for SSI encoder ... 159

O

OM ... 102
Operate modes ... 209
OR ... 119
Output for refrence adjustment ... 119
Outside window output ... 147
OW ... 147

P

PA ... 170
Parameter commands ... 38
PC ... 27
PD ... 190
PH ... 146
Phase Advance ... 170
Plus ... 181
PO ... 144, 146
Position control word ... 50
Position Encoder ... 202
Position from drive ... 190
Position overflow bound ... 108
Position overflow counter ... 109
Position trigger output ... 144
PR ... 20
Pretrigger for trace mode ... 167
Print hardware ... 20
Print help display ... 168
Print to terminal ... 20
Print version number ... 19
PS... ... 137
PT ... 47
PW ... 21

Q

QA ... 208

R

RA ... 119
RC... ... 114
Read input lines in byte g ... 124
Read output line states in byte g ... 125
Read SDO ... 189
Reference adjustment backwards output ... 120
Reference adjustment forwards output ... 120
Reference correction velocity ... 113
Reference error ... 161
Reference error adjustment acceleration ... 114
Reference error limit ... 112
reference filter options word ... 116
Reference holdoff time ... 119
Reference input ... 121
Reference input false high limit ... 117
Reference input false low limit ... 117
Reference input true high limit ... 117
Reference input true low limit ... 117
Reference options word ... 110

Reference position strange channel **120**
Reference repeat length **109**
Remainder **181**
Repeat command line **58**
RF... ..... **112**
RH **119**
RI... ..... **124**
RJ... ..... **115**
RK **120**
RL **109**
RM **109**
RN **115**
RO **125**
RP **58**
RR **148**
RS **24**
RT **95**
RV **113**
RW **110**

S

SA **39**
Safety **202**
Save parameters **22**
SB... ..... **108**
SC... ..... **41**
Scale mapping **78**
SE... ..... **94**
Send SDO **189**
Sequence commands **52**
Serial Communications **204**
Set acceleration **39**
Set acceleration feed-forward gain constant **100**
Set analogue control mode **174**
Set analogue control options word **176**
Set analogue control proportional gain **175**
Set analogue control setpoint **175**
Set analogue differential control gain **175**
Set analogue input high limit **178**
Set analogue input low limit **178**
Set analogue integral control gain **175**
Set autostart sequence **61**
Set auxiliary output channel **103**
Set backlash compensation distance **45**
Set baud rate **21**
Set baud rate modbus **21**
Set clutch length **86**
Set clutch time **85**
Set clutch window **86**
Set continous reference mode on/off **109**
Set control word **48**
Set creep distance **41**
Set deceleration **40**
Set deceleration for AB command **40**
Set deferred reference adjustment position **115**
Set differential gain constant **99**
Set error options word **96**
Set feedback encoder type **151**
Set filter on reference error **112**
Set increment select code **46**
Set input debounce time **147**
Set integral gain constant **99**
Set language **169**
Set map base advance **172**
Set map base offset for master map positions **77**
Set map base/offset/scale factor adjustment **79**
Set map base/offset/scale factor adjustment acceleration **80**
Set map link options word **83**
Set map options word **81**
Set map scale factor from bounds ratio **79**
Set maximum position error **94**
Set maximum reference correction **111**
Set modbus number **21**
Set monitor output function **101**
Set monitor output gain **101**
Set monitor output offset **102**
Set motor off error sequence **63**
Set number of bits for SSI encoder **159**
Set output line n in byte g **123**
Set password **21**
Set phase advance scale factor **170**
Set position **107**
Set position control word **50**
Set position limit high **94**
Set position limit low **95**
Set position overflow bound **108**
Set proportional gain constant **99**
Set reference correction velocity **113**
Set reference error adjustment acceleration **114**
Set reference error limit **112**
Set reference filter options word **116**
Set reference holdoff time **119**
Set reference input false high limit **117**
Set reference input false low limit **117**
Set reference input true high limit **117**
Set reference input true low limit **117**
Set reference offset **112**
Set reference options word **110**
Set reference repeat length **109**
Set reference timeout **95**
Set slave map positon offset **77**
Set slow speed **42**
Set slow velocity mode **43**
Set system prompts **169**
Set timeout **94**
Set timeout for window **44**
Set velocity **38**
Set velocity averaging time constant **172**
Set velocity feedback gain constant **99**
Set virtual motor mode **27**
Set wait status variable **188**
Set window **43**
Set/display position overflow counter **109**
SetReferenceAdvanceFactor **115**

SF ... 101
SK ... 26
Slave map positon offset ... 77
Slow speed ... 42
Slow velocity mode ... 43
SM ... 78
Snapshot position data ... 162
SO ... 123
software license key ... 26
Software updaten ... 23
SP ... 22, 23
SR... ... 111
SRAM ... 184
SS ... 42
SSI ... 151
ST ... 32
Status Codes ... 200
Status messages ... 191
Stop command ... 32
Store variable in SRAM ... 184
SV ... 38
SW ... 43
SY ... 169
Syntax ... 11
System prompts ... 169

T

TC ... 130
TD ... 168
***tendo*®** ... 24
TF ... 167
TG ... 134
TH ... 166
TI ... 44
Time ... 162
Time step to trace mode ... 164
Timeout for window ... 44
TK ... 133
TL ... 166
TM ... 76
TO ... 94
TP ... 167
TR ... 164
Trace mode ... 164
Transfer map data ... 76
Trapezoidal profile ... 28
Triggerchannel for trace mode ... 166
Triggerlevel for trace mode ... 166
Triggertransition for trace mode ... 167
TT ... 164
TW ... 165
TZ ... 134

U

UI ... 129
UL ... 77
Undefine input function ... 129
Undefine output function ... 129
Unlink slave axis from master axes ... 77
UO ... 129
Update ... 23

V

Variable assignment ... 182
Variable commands ... 179
Variable map number assignment ... 182
Variable number program assignment ... 182
VC ... 34
Velocity ... 161
Velocity averaging time constant ... 172
velocity feedback ... 98
velocity feed-forward ... 98
Velocity trigger output ... 146
VH ... 147
Virtual In- and Outputs ... 122
Virtual motor mode ... 27
VJ ... 43
VM ... 27
VN ... 19
VT ... 172
VX ... 185

W

WA ... 89
Wait for absolute position ... 89
Wait for baund position ... 91
Wait for bound overflow count ... 92
Wait for input line ... 88
Wait for motor status ... 91
Wait for reference input ... 91
Wait for relativ position ... 90
Wait for time ... 87
Wait for write to variable ... 183
Wait status variable ... 188
WB ... 91
WC ... 92
WE ... 92
WF ... 91
WI ... 88
WR ... 90
Wraparound offset value ... 162
WS ... 91
WT ... 87
WV ... 183

X

XA ... 40
XM ... 74
XP ... 57

XR **178**
XS **57**
XV **75**
XX **75**

Z

ZC **107**
Zero position counter or set position **107**
ZH **117**
ZL **117**
ZT **25**
ZW **50**
ZX **118**
ZY **118**

...
In many countries we are represented by our subsidiaries and sales partners.

Please refer to our Homepage for further details or contact our headquarters.

www
www.pilz.com

Technical support
+49 711 3409-444

Pilz GmbH & Co. KG
Sichere Automation
Felix-Wankel-Straße 2
73760 Ostfildern, Germany
Telephone: +49 711 3409-0
Telefax: +49 711 3409-133
E-Mail: pilz.gmbh@pilz.de

pilz
more than automation
safe automation

21 507-07, 2005-11 Printed in Germany